



Universidad Politécnica de Madrid
Facultad de Informática

Doctoral Thesis

Benchmarking Semantic Web technology

Author: Raúl García Castro

Advisor: Prof. Dr. Asunción Gómez Pérez

July 2008

A los padres
A mi abuela María

Agradecimientos

La elaboración de esta tesis a lo largo de los últimos años me ha costado mucho sudor, pero todas las lágrimas han brotado en la escritura de esta página. Lo primero ha sido muy fácil comparado con lo segundo.

Esta tesis está dedicada a Tomás, mi padre, quien me ha hecho ser como soy, pensar como pienso y luchar como lucho.

Me he esforzado en ella, aún sabiendo que él no podrá ver los resultados, pero intentando que se sintiera orgulloso de ellos y de mí mismo.

Además, tengo que agradecer a Dácil y a toda mi familia (a todas mis familias), a los que vienen y a los que se van, toda la comprensión, el cariño y el apoyo mostrado no sólo durante estos años, sino durante toda mi vida.

No obstante, sacar adelante una tesis no es un trabajo solitario. Antes que a nadie debo nombrar a Asun, por todo lo que me ha enseñado, a Charo, por haber convertido mis textos ilegibles en textos legibles en correcto inglés, y a Holger, por toda su ayuda y buenos consejos.

También quiero mencionar a toda la gente del grupo, ya que sin ellos no se aguantarían ni las cosas buenas ni las malas del día a día, y a todos aquellos con los que he trabajado en estos últimos años.

Finalmente, quiero dar gracias a toda la gente que me ha ayudado, ya sea en mayor o menor medida, en las actividades de benchmarking realizadas en esta tesis: la gente con la que he trabajado en el grupo: Stefano, Jesús, Silvia y Moisés; todos aquellos que participaron en el RDF(S) Interoperability Benchmarking: Olivier Corby, York Sure, Moritz Weiten y Markus Zondler; y los que participaron en el OWL Interoperability Benchmarking: Stamatia Dasiopoulou, Danica Damjanovic, Michael Erdmann, Christian Fillies, Roman Korf, Diana Maynard, York Sure, Jan Wielemaker, y Philipp Zaltenbach. Sin su esfuerzo esto no habría sido posible.

Madrid, mayo de 2008

Abstract

Semantic Web technologies need to interchange ontologies for further use. Due to the heterogeneity in the knowledge representation formalisms of the different existing technologies, interoperability is a problem in the Semantic Web and the limits of the interoperability of current technologies are yet unknown.

A massive improvement of the interoperability of current Semantic Web technologies, or of any other characteristic of these technologies, requires continuous evaluations that should be defined and conducted in consensus, using generic, reusable, freely-available, and affordable tools and methods.

This thesis presents the following contributions to the field of benchmarking within Semantic Web technologies:

- It proposes a benchmarking methodology for Semantic Web technologies.
- It defines the UPM Framework for Benchmarking Interoperability, an evaluation infrastructure that includes all the resources (experiment definitions, benchmark suites and tools) needed for benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.
- It describes two interoperability benchmarking activities carried out over Semantic Web technologies and provides detailed interoperability results of the tools that participated in them; the RDF(S) Interoperability Benchmarking that contemplates interoperability using RDF(S) as the interchange language, and the OWL Interoperability Benchmarking that contemplates interoperability using OWL as the interchange language.

Resumen

Las diferentes tecnologías de la Web Semántica necesitan intercambiar ontologías para su posterior utilización. La Web Semántica, por otro lado, tiene que hacer frente al problema de la interoperabilidad, que está causado, en gran medida, por la heterogeneidad de los formalismos de representación de conocimiento de las distintas tecnologías existentes, siendo la interoperabilidad en la Web Semántica un problema cuyos límites hoy se desconocen.

Una mejora masiva de la interoperabilidad de las tecnologías actuales de la Web Semántica, o de cualquier otra característica de dichas tecnologías, requiere evaluaciones continuas, que sean definidas y realizadas en consenso, utilizando herramientas y métodos que sean genéricos, reutilizables, públicos y económicos.

Esta tesis presenta las siguientes contribuciones al campo de *benchmarking* en las tecnologías de la Web Semántica:

- Propone una metodología de *benchmarking* para las tecnologías de la Web Semántica.
- Define el *UPM Framework for Benchmarking Interoperability*, una infraestructura de evaluación que incluye todos los recursos (definiciones de experimentos, conjuntos de pruebas y herramientas) necesarios para hacer *benchmarking* de la interoperabilidad de las tecnologías de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio.
- Describe dos actividades de *benchmarking* de las tecnologías de la Web Semántica y ofrece los resultados detallados de las herramientas que participaron en las mismas; el *RDF(S) Interoperability Benchmarking*, que contempla la interoperabilidad utilizando RDF(S) como lenguaje de intercambio, y el *OWL Interoperability Benchmarking*, que también contempla la interoperabilidad pero utiliza OWL como lenguaje de intercambio.

Contents

1. Introduction	1
1.1. Context	1
1.1.1. The Semantic Web	1
1.1.2. Brief introduction to Semantic Web technologies	3
1.1.3. Semantic Web technology evaluation	5
1.2. The need for benchmarking in the Semantic Web	9
1.3. Semantic Web technology interoperability	10
1.3.1. Heterogeneity in ontology representation	12
1.3.2. The interoperability problem	13
1.3.3. Categorising ontology differences	14
1.4. Thesis contributions	15
1.5. Thesis structure	16
2. State of the Art	19
2.1. Software evaluation	19
2.2. Benchmarking	23
2.2.1. Benchmarking vs evaluation	23
2.2.2. Benchmarking classifications	24
2.3. Evaluation and improvement methodologies	25
2.3.1. Benchmarking methodologies	26
2.3.2. Software Measurement methodologies	31
2.3.3. Experimental Software Engineering methodologies	34
2.4. Benchmark suites	41
2.5. Previous interoperability evaluations	43
2.6. Conclusions	45
3. Work objectives	47
3.1. Thesis goals and open research problems	47
3.2. Contributions to the state of the art	48
3.3. Work assumptions, hypothesis and restrictions	51

4. Benchmarking methodology for Semantic Web technologies	55
4.1. Design principles	56
4.2. Research methodology	58
4.2.1. Selection of relevant processes	58
4.2.2. Identification of the main tasks	59
4.2.3. Task adaption and completion	60
4.2.4. Analysis of task dependencies	62
4.3. Benchmarking methodology	62
4.3.1. Benchmarking actors	62
4.3.2. Benchmarking process	63
4.3.3. Plan phase	64
4.3.4. Experiment phase	70
4.3.5. Improvement phase	72
4.3.6. Recalibration task	76
4.4. Organizing the benchmarking activities	76
4.4.1. Plan phase	76
4.4.2. Experiment phase	84
5. RDF(S) Interoperability Benchmarking	87
5.1. Experiment definition	88
5.1.1. RDF(S) Import Benchmark Suite	89
5.1.2. RDF(S) Export Benchmark Suite	96
5.1.3. RDF(S) Interoperability Benchmark Suite	99
5.2. Experiment execution	102
5.2.1. Experiments performed	103
5.2.2. Experiment automation	103
5.3. RDF(S) import results	103
5.3.1. KAON RDF(S) import results	104
5.3.2. Protégé-Frames RDF(S) import results	105
5.3.3. WebODE RDF(S) import results	107
5.3.4. Corese, Jena and Sesame RDF(S) import results	109
5.3.5. Evolution of RDF(S) import results	110
5.3.6. Global RDF(S) import results	115
5.4. RDF(S) export results	117
5.4.1. KAON RDF(S) export results	117
5.4.2. Protégé-Frames RDF(S) export results	119
5.4.3. WebODE RDF(S) export results	120
5.4.4. Corese, Jena and Sesame RDF(S) export results	122
5.4.5. Evolution of RDF(S) export results	123
5.4.6. Global RDF(S) export results	127
5.5. RDF(S) interoperability results	129
5.5.1. KAON interoperability results	129
5.5.2. Protégé-Frames interoperability results	132
5.5.3. WebODE interoperability results	135
5.5.4. Global RDF(S) interoperability results	138

6. OWL Interoperability Benchmarking	143
6.1. Experiment definition	144
6.2. The OWL Lite Import Benchmark Suite	146
6.2.1. Benchmarks that depend on the knowledge model	147
6.2.2. Benchmarks that depend on the syntax	147
6.2.3. Description of the benchmarks	150
6.2.4. Towards benchmark suites for OWL DL and Full	152
6.3. Experiment execution: the IBSE tool	154
6.3.1. IBSE requirements	155
6.3.2. IBSE implementation	156
6.3.3. Using IBSE	162
6.4. OWL compliance results	163
6.4.1. GATE OWL compliance results	164
6.4.2. Jena OWL compliance results	167
6.4.3. KAON2 OWL compliance results	167
6.4.4. Protégé-Frames OWL compliance results	170
6.4.5. Protégé-OWL OWL compliance results	173
6.4.6. SemTalk OWL compliance results	174
6.4.7. SWI-Prolog OWL compliance results	177
6.4.8. WebODE OWL compliance results	178
6.4.9. Global OWL compliance results	182
6.5. OWL interoperability results	184
6.5.1. OWL interoperability results per tool	184
6.5.2. Global OWL interoperability results	195
6.6. Evolution of OWL interoperability results	201
6.6.1. OWL compliance results	203
6.6.2. OWL interoperability results	208
7. Conclusions and future research lines	211
7.1. Development and use of the benchmarking methodology	211
7.2. Benchmarking interoperability	215
7.3. RDF(S) and OWL interoperability results	218
7.4. Open research problems	221
7.5. Dissemination of results	223
Bibliography	225
A. Combinations of the RDF(S) components	235
A.1. Benchmarks with single components	236
A.2. Benchmarks with combinations of two components	236
A.3. Benchmarks with combinations of more than two components	238
B. Description of the RDF(S) benchmark suites	241
B.1. RDF(S) Import Benchmark Suite	241
B.2. RDF(S) Export and Interoperability Benchmark Suites	255

C. Combinations of the OWL Lite components	269
C.1. Benchmarks for classes	269
C.2. Benchmarks for properties	272
C.3. Benchmarks for instances	274
D. The OWL Lite Import Benchmark Suite	279
D.1. List of benchmarks	279
D.2. Description of ontologies in DL	295
E. The IBSE ontologies	305
F. Resumen amplio en español	309

List of Tables

2.1. Relation between the state of the art and the contributions. . . .	20
2.2. Comparison of tasks in benchmarking methodologies.	30
2.3. Common tasks in benchmarking methodologies.	31
2.4. Common tasks in Software Measurement methodologies.	34
2.5. Comparison of tasks in Software Measurement methodologies. . .	35
2.6. Comparison of tasks in Experimental S. E. methodologies. . . .	40
2.7. Common tasks in Experimental S. E. methodologies.	41
2.8. Experiments carried out at the EON2003 workshop.	44
4.1. Common tasks identified in the relevant processes.	61
4.2. Ontology development tools able to import/export RDF(S). . . .	81
4.3. Ontology development tools able to import/export OWL.	82
4.4. Tools participating in the RDF(S) Interoperability Benchmarking.	83
4.5. Tools participating in the OWL Interoperability Benchmarking. .	83
5.1. Groups of the RDF(S) import benchmarks.	92
5.2. An example of an RDF(S) import benchmark definition.	93
5.3. Fictitious results of executing the benchmark I46.	94
5.4. Groups of the RDF(S) export benchmarks.	98
5.5. An example of an RDF(S) export benchmark definition.	98
5.6. Summary of KAON's RDF(S) import result evolution.	111
5.7. Summary of Protégé's RDF(S) import result evolution.	112
5.8. Summary of WebODE's RDF(S) import result evolution.	113
5.9. Summary of KAON's RDF(S) export result evolution.	124
5.10. Summary of Protégé's RDF(S) export result evolution.	125
5.11. Summary of WebODE's RDF(S) export result evolution.	126
5.12. RDF(S) interoperability results from all the tools to KAON. . . .	130
5.13. RDF(S) interoperability results from all the tools to Protégé. . .	133
5.14. RDF(S) interoperability results from all the tools to WebODE. .	135
5.15. Combinations of components interchanged between the tools. . .	141
6.1. An example of an OWL import benchmark definition.	150
6.2. Groups of the OWL import benchmarks.	151
6.3. Restrictions in the use of OWL Lite and OWL DL.	153

6.4. Results in <i>Step 1</i> (for 82 benchmarks).	182
6.5. Percentage of identical ontologies per group in <i>Step 1</i> .	183
6.6. Subgroups of the OWL import benchmarks.	185
6.7. OWL interoperability results of GATE.	187
6.8. OWL interoperability results of Jena.	188
6.9. OWL interoperability results of KAON2.	189
6.10. OWL interoperability results of Protégé-Frames.	190
6.11. OWL interoperability results of Protégé-OWL.	191
6.12. OWL interoperability results of SemTalk.	192
6.13. OWL interoperability results of SWI-Prolog.	193
6.14. OWL interoperability results of WebODE.	194
6.15. Percentage of identical interchanged ontologies.	195
6.16. Percentage of identical interchanged ontologies for <i>Group A</i> .	196
6.17. Percentage of identical interchanged ontologies for <i>Group B</i> .	197
6.18. Percentage of identical interchanged ontologies for <i>Group C</i> .	197
6.19. Percentage of identical interchanged ontologies for <i>Group D</i> .	197
6.20. Percentage of identical interchanged ontologies for <i>Group E</i> .	198
6.21. Percentage of identical interchanged ontologies for <i>Group F</i> .	198
6.22. Percentage of identical interchanged ontologies for <i>Group G</i> .	198
6.23. Percentage of identical interchanged ontologies for <i>Group H</i> .	199
6.24. Percentage of identical interchanged ontologies for <i>Group I</i> .	199
6.25. Percentage of identical interchanged ontologies for <i>Group J</i> .	199
6.26. Percentage of identical interchanged ontologies for <i>Group K</i> .	200
6.27. Percentage of benchmarks in which tool execution fails in <i>Step 1</i> .	200
6.28. Percentage of benchmarks in which tool execution fails in <i>Step 2</i> .	200
6.29. Percentage of identical interchanged ontologies per group.	202
6.30. Updated results in <i>Step 1</i> (for 82 benchmarks).	203
6.31. Updated percentage of identical ontologies per group in <i>Step 1</i> .	204
6.32. Updated percentage of identical interchanged ontologies.	208
6.33. Updated OWL interoperability results of WebODE.	210
D.1. Description Logics notation from [Volz, 2004].	295
D.2. Sample ontology description in the Description Logics formalism.	295

List of Figures

1.1. The Semantic Web architecture.	2
1.2. Example of ontology interchanges in the Semantic Web.	11
1.3. Knowledge models of Protégé-Frames, WebODE and RDF(S).	12
1.4. Ontology interchanges within Semantic Web tools.	13
1.5. Classification of ontology heterogeneity levels [Barrasa, 2007].	15
2.1. Quality model for internal and external quality [ISO/IEC, 2001].	22
2.2. Quality model for quality in use [ISO/IEC, 2001].	23
2.3. Benchmarking benefits.	24
3.1. The UPM Framework for Benchmarking Interoperability.	49
4.1. Steps followed during the development of the methodology.	59
4.2. The benchmarking process.	63
4.3. <i>Plan</i> phase of the benchmarking process.	65
4.4. <i>Experiment</i> phase of the benchmarking process.	71
4.5. <i>Improvement</i> phase of the benchmarking process.	73
5.1. UPM-FBI resources for benchmarking RDF(S) interoperability.	87
5.2. Evaluations performed in the RDF(S) experiments.	88
5.3. Procedure for executing an RDF(S) import benchmark.	95
5.4. Procedure for executing an RDF(S) export benchmark.	100
5.5. Procedure for executing an RDF(S) interoperability benchmark.	102
5.6. Evolution of the RDF(S) import results in KAON.	111
5.7. Evolution of the RDF(S) import results in Protégé.	112
5.8. Evolution of the RDF(S) import results in WebODE.	113
5.9. Final RDF(S) import results.	115
5.10. Evolution of the RDF(S) export results in KAON.	124
5.11. Evolution of the RDF(S) export results in Protégé.	125
5.12. Evolution of the RDF(S) export results in WebODE.	126
5.13. Final RDF(S) export results.	128
5.14. RDF(S) interoperability results from all the tools to KAON.	138
5.15. RDF(S) interoperability results from all the tools to Protégé.	139
5.16. RDF(S) interoperability results from all the tools to WebODE.	139

6.1.	UPM-FBI resources for benchmarking OWL interoperability. . .	143
6.2.	The two steps of a benchmark execution.	145
6.3.	The OWL DL Import Benchmark Suite.	153
6.4.	Automatic experiment process in IBSE.	156
6.5.	Graphical representation of the <i>benchmarkOntology</i> ontology. . .	157
6.6.	Graphical representation of the <i>resultOntology</i> ontology.	158
6.7.	Implementation of the <i>ImportExport</i> method for Jena.	160
6.8.	OWL import and export operation results for GATE.	165
6.9.	OWL import and export operation results for Jena.	167
6.10.	OWL import and export operation results for KAON2.	168
6.11.	OWL import and export operation results for Protégé-Frames. . .	170
6.12.	OWL import and export operation results for Protégé-OWL. . .	174
6.13.	OWL import and export operation results for SemTalk.	175
6.14.	OWL import and export operation results for SWI-Prolog.	178
6.15.	OWL import and export operation results for WebODE.	179
6.16.	Updated OWL import and export operation results for WebODE. .	203
A.1.	The components of the RDF(S) knowledge model.	235
B.1.	Notation used in the RDF(S) Import Benchmark Suite figures. .	241
B.2.	Notation used in the RDF(S) Export Benchmark Suite figures. .	255
D.1.	Notation used in the OWL Lite Import Benchmark Suite figures. .	280

Introduction

1.1. Context

1.1.1. The Semantic Web

The World Wide Web (also known as the “WWW” or “Web”) is the universe of network-accessible information, the embodiment of human knowledge¹. The Web has been built on a body of software, and a set of protocols and conventions, which makes it easy for anyone, through the use of hypertext and multimedia techniques, to roam, browse, and contribute to it.

Although information on the Web was intended to be useful and accessible both for humans and machines, most of it has been designed for human consumption; so, computer programs find difficult to manipulate the Web meaningfully and to process its semantics. The Semantic Web, on the other hand, upraised not as a separate Web but as an extension of the current one in which information is given well-defined meaning, better enabling computers and people to work in cooperation [Berners-Lee *et al.*, 2001]. Nowadays, the Semantic Web is a web of data and information that provides common formats for representing knowledge and inference rules, allowing the aggregation and combination of data drawn from different resources.

Figure 1.1 shows the different layers of the Semantic Web architecture in its last version². Up to now, standardization efforts have focused on the lower layers of this architecture, which are next described. XML³ provides a basic format for structured documents, RDF⁴, a format for representing data, RDF-S⁵, data typing and allows document structure to be constrained, OWL⁶, a language to represent ontologies that allows more powerful schemas, SPARQL⁷, a language for executing queries, and finally, the rule layer allows the representation of inference rules. The upper layers deal with the specification of a logical language

¹<http://www.w3.org/WWW/>

²<http://www.w3.org/2001/sw/>

³<http://www.w3.org/XML/>

⁴<http://www.w3.org/RDF/>

⁵<http://www.w3.org/TR/rdf-schema/>

⁶<http://www.w3.org/2004/OWL/>

⁷<http://www.w3.org/TR/rdf-sparql-query/>

that has inference and functions and which is powerful enough to be able to define the rest; it also deals with a proof language that allows sending assertions, together with the inference path leading to an assertion from assumptions made, and with digital signatures that can be used to verify that the attached information has been provided by a specific trusted source.

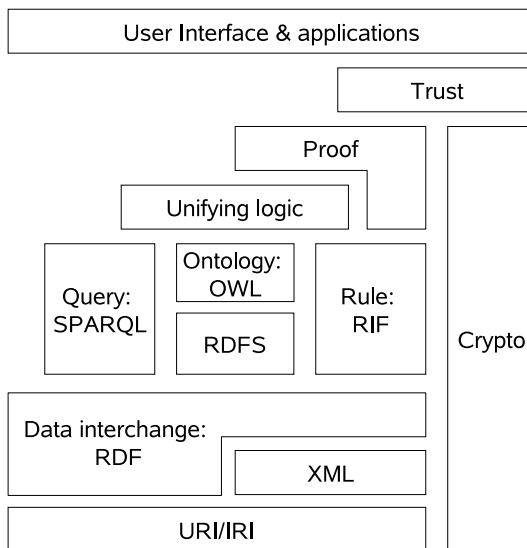


Figure 1.1: The Semantic Web architecture.

Some people perceive the Semantic Web as part of the Artificial Intelligence field, but the Semantic Web is not only Artificial Intelligence. Artificial Intelligence aims to make machines that simulate human intelligence, while the Semantic Web is an initiative for computers and people whose goal is to create a universal medium for the exchange of data. Although the Semantic Web uses technologies from Artificial Intelligence, it also uses technologies from other computer science fields (Software Engineering, databases, programming languages, communications, etc.).

To help make the Semantic Web possible, the Artificial Intelligence field offers two substantial pillars, Knowledge Representation and inference techniques, which have been deeply studied by Artificial-Intelligence researchers. The former provides the Semantic Web with languages for expressing domain models (also known as ontologies) and data in a machine processable form, aiming at a machine-understandable Web. The latter provides the Semantic Web with inference rules both for automated reasoning about this data, using the domain models within a restricted framework, and for inferring information that is not explicitly expressed.

1.1.2. Brief introduction to Semantic Web technologies

This section presents an overview of the different types of technologies⁸ that support the different layers of the Semantic Web architecture, shown in figure 1.1.

Our focus is not on the technologies used in the lower layers of the architecture (URI, Unicode and XML) but on the technologies developed to manage semantic information (data, ontologies and rules) and to query in the middle layers. Even though these layers are separated, in some cases technologies manage information from more than one layer (e.g. ontologies and data or ontologies and rules).

Although Semantic Web technologies are highly heterogeneous in use and purpose, they are frequently used conjointly for performing tasks in different stages of the ontology lifecycle, ranging from the development, use and maintenance of ontologies to the development of semantic applications.

Moreover, these technologies need a different degree of human intervention. They can be executed manually, semi-automatically and automatically, and these technologies provide different interfaces for accessing them, like user interfaces, programming interfaces, protocols, or services.

The classification presented below arranges Semantic Web technologies into groups according to the use of the semantic information. It has been elaborated from different Semantic Web technology classifications [Gómez-Pérez *et al.*, 2003, Davies *et al.*, 2006, García-Castro *et al.*, 2007d] and from other Semantic Web technology classifications found on the Web^{9,10,11,12}. We can add that new Semantic Web tools appear every day and that these sources contain updated information with many examples of tools in each category.

The Semantic Web technology groups¹³ identified are the following:

- **Ontology development.** This category includes two types of tools: *ontology editors*, which support ontology development, and *ontology learning tools*, which generate ontologies from natural language texts and semi-structured sources and databases. These tools are frequently used with other tools or inside ontology development environments.
- **Ontology management.** Several types of tools are included in this category: *ontology evaluation tools* and *ontology validation tools*, which evaluate ontologies according to some user criteria or some specification, respectively; *ontology evolution tools*, which manage the ontology evolu-

⁸Sometimes it can be seen in the literature that the Semantic Web specifications (RDF, OWL, etc.) are called technologies, but in this thesis the term technology is used to refer to software applications or tools.

⁹<http://esw.w3.org/topic/SemanticWebTools>

¹⁰<http://planetrdf.com/guide/>

¹¹http://www.mkbergman.com/?page_id=346

¹²<http://deswap.informatik.hu-berlin.de/>

¹³These groups are not described thoroughly because the previous studies already included detailed information and the definitions used here.

tion and versioning; and *ontology alignment tools*, which create and use ontology alignments for merging, translating or transforming ontologies.

- **Instance generation.** These tools are used for generating instances according to some ontologies, and they include the following: *instance editors*, which support the manual edition of instances; *ontology populators*, which generate automatically instances in an ontology from a data source; and *ontology-based annotation tools*, which annotate manually or automatically multimedia documents (text, images, audio, video, etc.) with metadata according to some ontology and, therefore, they define ontology instances.
- **Semantic information storage.** This category includes tools for persistent information storage and they can be divided into three different types: *ontology repositories*, which store ontologies; *data repositories*, which store data; and *metadata registries*, which store metadata. Additionally, these tools usually provide querying and inferencing functionalities.
- **Querying and reasoning.** These tools generate and process queries over ontologies. Such tools can be divided into three types: *query editors*, which support query creation; *query processors*, which manage query answering over ontologies in distributed sources (e.g., translating queries and their results from one ontology to another or merging results from different sources); and *reasoners*, which perform reasoning tasks over an ontology, such as consistency, subsumption, or satisfiability checking.
- **Semantic information access.** These tools provide functionalities to search and access semantic information. These are the following: *searching and browsing tools*, which search and browse semantic information; *visualization tools*, which adapt views of semantic information to fit a particular purpose; and *ontology customization tools*, which adapt ontologies according to some user needs.
- **Programming and development.** These include *programming environments* that help develop applications that use Semantic Web information. They can be either specific to one implementation language or valid for multiple implementation languages.
- **Application integration.** These tools aim to integrate applications on the Web by using Semantic Web services (Web services with semantic descriptions) and to automate the execution of tasks such as the discovery, negotiation, composition, or invocation of these services.

This classification does not cover exhaustively all the existing technologies used in the Semantic Web, additionally other technologies that are not specific to the Semantic Web are also used, for example, Human Language Technologies, which are highly employed in ontology learning and population tasks.

1.1.3. Semantic Web technology evaluation

The widespread use of the Semantic Web depends on the two types of evaluation made in this area: evaluation of the technologies that use the content of the Semantic Web (Semantic Web technology evaluation) and evaluation of the content itself (ontology evaluation) [Sure *et al.*, 2004]. In this thesis, evaluation is only considered in terms of technology evaluation.

Semantic Web technologies have to be evaluated like any other software technology, and its evaluation shares the same principles as any other software evaluation, but with different viewpoints and objectives. However, this does not mean that Semantic Web technology evaluations have to be performed from scratch, because we can find extensive methods and tools to perform software evaluations both in the Software Engineering literature [Sommerville, 2006] and in the Experimental Software Engineering one [Wohlin *et al.*, 2000, Boehm *et al.*, 2005].

Traditional software evaluation approaches can (and should) be used for evaluating Semantic Web technologies, but they do not suffice for evaluating Semantic Web technologies since they do not cover the specific characteristics and uses of these technologies such as the use of ontologies as data models, the assumption about the incompleteness of the system information (open world assumption), the inference of new information, or the use of the W3C standards presented at the beginning of the chapter. Therefore, new evaluation methods, infrastructures and metrics have to be defined for Semantic Web technologies to validate research results and to show the benefits of these technologies.

However, in the Semantic Web area technology evaluation is seldom carried out [Sure *et al.*, 2004], even though several community efforts have appeared in the form of evaluation and benchmarking activities and evaluation-related workshops. Next, those efforts that have made the highest impact in the community are presented:

- **The deliverable 1.3** [OntoWeb, 2002] of the OntoWeb European Thematic Network (IST-2000-29243)¹⁴ presented a general framework for comparing ontology-related technologies. This framework identified the following types of tools: ontology building tools, ontology merge and integration tools, ontology evaluation tools, ontology based annotation tools, and ontology storage and querying tools. For each type of tool, the framework provided a set of qualitative criteria (such as the software platform where the tool runs, the knowledge representation language that the tool manages, the use of inference services, the existence of documentation, or the usability of the tool) for comparing tools in each group as well as a theoretical comparison of different tools in each group.
- **The Ontology Alignment Evaluation Initiative**¹⁵ (OAEI) is an international initiative that has been running since 2004 and has organized five ontology alignment contests in different workshops with the goal of

¹⁴<http://www.ontoweb.org/>

¹⁵<http://oaei.ontologymatching.org/>

establishing a consensus for evaluating ontology alignment methods and their associated tools.

In these contests, ontology alignment systems are compared over a common set of synthetic and real-world tests using a common evaluation framework. The metrics to evaluate the alignments produced by the tools are precision and recall, although some researchers also use other measures derived from these, such as aggregations of precision and recall (e.g., f-measure) or generalisations of precision and recall (e.g., symmetric, effort-based, precision-oriented and recall-oriented proximities).

In 2004 two alignment contests took place within the Information Interpretation and Integration Conference (I3CON2004) and within the Evaluation of Ontology-based Tools workshop (EON2004). In 2005, the alignment contest took place within the Integrating Ontologies workshop (IntOnt2005) and in 2006 and 2007 within the Ontology Matching workshops (OM2006 and OM2007 respectively).

- **The RDF(S) and OWL Interoperability Benchmarkings**¹⁶ presented in this thesis ran from 2005 to 2006 and from 2006 to 2007 respectively. They involved evaluating and improving the interoperability of different types of Semantic Web technologies using RDF(S) and OWL as interchange languages.
- **The Semantic Web Service Challenge**¹⁷ (SWS Challenge) is an international initiative that has been running since 2006 and has organized six challenges in different workshops. The goal of the SWS Challenge is, on the one hand, to explore trade-offs among existing approaches for automating the mediation, choreography and discovery of Web Services using semantic annotations and, on the other hand, to reveal the strengths and weaknesses of the proposed approaches and those aspects of the problem space not yet covered.

In 2006, three workshops took place, the first one was an independent workshop, whereas the other two were held within the European Semantic Web Conference (ESWC2006) and the International Semantic Web Conference (ISWC2006). In 2007, another three workshops were held within the European Semantic Web Conference (ESWC2007), within the International Conference on Enterprise Information Systems (ICEIS2007) and within the IEEE/WIC/ACM International Conference on Web Intelligence (WI 2007).

The SWS Challenge proposes evaluating functionality rather than performance. In this contest, technologies are tested over a set of common problems and their functionality is certified by the workshop participants through a peer-review process.

¹⁶http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

¹⁷<http://sws-challenge.org/>

The problems proposed build upon an initial mediation problem, adding further levels of mediation and discovery problems on top it, each one corresponding to a general kind of problem with sublevels of complexity.

Then, the results of a system for a problem are classified into five levels of success: 1) the system does not invoke the requested web services, 2) the system adequately invokes the web services (level 0), 3) the code of the system has to be changed to solve the problem (level 1), 4) only data has to be changed (level 2), and 5) the system does not undergo any changes at all (level 3). On the other hand, level 0 is minimal and is automatically determined by the system, whereas the following three levels are determined by peer review. A higher success level indicates a better solution to the problem.

- **The Evaluation of Ontologies and Ontology-based Tools international workshops** (EON) have been taking place since 2002; their 5th edition was held in 2007¹⁸.

The participants in the first EON workshop (EON2002) conducted an experiment that consisted in modelling a tourism domain ontology in different ontology development tools and then exporting the modelled ontologies to a common language (RDF(S)). The goal of this experiment was to analyse the modelling decisions, limitations and problems that should be contemplated when dealing with these tools.

In the second EON workshop (EON2003), the experiment proposed aimed to evaluate the import, export and interoperability of different ontology building tools using an interchange language. This experiment was performed by exporting and importing ontologies to an intermediate language and assessing the amount of knowledge lost during these transformations. Some experiments evaluated export functionalities, others, import functionalities and only a few evaluated interoperability. However, no systematic evaluation was performed since each experiment used different evaluation procedures, different interchange languages (DAML, RDF(S), OWL, and UML were used), and different principles for modelling ontologies.

The third EON workshop (EON2004) included one of the OAEI contests mentioned before and targeted the characterization of alignment methods with regard to a common evaluation framework.

In the fourth EON workshop (EON2006) the topic was changed from evaluation of ontology technologies to evaluation of ontologies and, consequently, none of the papers presented were related to ontology technology evaluation.

In the fifth EON workshop (EON2007) the topic was changed again but now both topics were covered, ontology evaluation and ontology technology evaluation. This time, however, the workshop did not propose any community technology evaluation.

¹⁸<http://km.aifb.uni-karlsruhe.de/ws/eon2007>

- **The Scalable Semantic Web Knowledge Base Systems international workshops (SSWS)** have been held since 2005 and their 3rd edition took place in 2007¹⁹. The main topic of these workshops is the scalability of Semantic Web technologies and optimization methods and techniques for building scalable knowledge base systems for the Semantic Web. These workshops, on the other hand, do not propose community evaluations and the technology evaluations presented in them are mainly focused on validating the proposed optimization approaches.
- **The International and European Semantic Web Conferences** (ISWC and ESWC, respectively) usually include Semantic Web technology evaluation sessions. These sessions are held in the ISWC since 2003, but in 2006 one session took place in the ESWC.

In addition to the mentioned evaluation and benchmarking activities, there are a set of benchmark suites that have been used at large in the whole Semantic Web community. These are the following:

- **The RDF and OWL Test Cases.** In the scope of the W3C, the RDF Test Cases²⁰ [Grant and Beckett, 2004] and the OWL Test Cases²¹ [Carroll and Roo, 2004] were created by the W3C RDF Core Working Group and the W3C Web Ontology Working Group, respectively.

The RDF and OWL Test Cases check the correctness of the tools that implement RDF and OWL knowledge bases. They are also intended to provide examples for, and clarification of, the normative definition of the languages and to illustrate the resolution of different issues considered by the Working Groups.

- **The Lehigh University Benchmark.** The Lehigh University Benchmark (LUBM) [Guo *et al.*, 2003, Guo *et al.*, 2005] can be used to evaluate systems with different reasoning capabilities and storage mechanisms.

This benchmark features an ontology for the university domain and synthetic OWL data that can be scaled to an arbitrary size; it also boasts fourteen extensional queries representing a variety of properties (input size, selectivity, complexity, assumed hierarchy information, and assumed inference) and several performance metrics (load time, size after loading, query answering time, completeness and soundness regarding the queries, and a combined metric for query answering time and answer completeness and soundness).

The LUBM has been widely used to evaluate reasoners, and some extensions of it have been proposed. One of such is the University Ontology Benchmark (UOBM) [Ma *et al.*, 2006], which extends the expressiveness of the LUBM ontology to make a thorough use of OWL Lite and OWL

¹⁹<http://www.cs.rmit.edu.au/fedconf/index.html?page=ssws2007cfp>

²⁰<http://www.w3.org/TR/rdf-testcases/>

²¹<http://www.w3.org/TR/owl-test/>

DL and creates interrelations between the synthetic data. Weithöner et al. [2007] propose a new set of benchmarks that focus on specific issues disregarded in the LUBM: the influence of the ontology complexity on instance reasoning, the effects of the OWL serialization used, and the effects of using previously computed implicit knowledge or results.

In summary, in the last few years the number of evaluation and benchmarking activities has been continuously increasing within the Semantic Web area. Nevertheless, Semantic Web technologies have not been thoroughly evaluated and the number of evaluations in this area is still not high enough to ensure a high quality technology.

However, this is not a negative assertion since current Semantic Web technologies are mainly being developed in research institutions, which implies that technology evaluations are punctual, focused on validating research results, not documented enough in the research papers where they are normally described, usually performed by one person or organization, and executed in particular settings. Furthermore, these evaluations deal with a small group of Semantic Web tools (mainly ontology alignment tools, ontology development tools, ontology repositories, and reasoners) and are not applicable, in general, to other types of tools.

These facts make the evaluation of Semantic Web technologies difficult and expensive and set up an important barrier that prevents the easy transference of such technologies to the market, especially today when companies are reusing and developing Semantic Web technologies and when companies based on Semantic Web technologies and providers of Semantic Web services are starting off.

Another important problem is that most people do not know how to evaluate Semantic Web technologies. Besides, it is difficult to reuse the evaluation results provided by third parties and the lessons they learnt; therefore, new evaluation methods and tools have to be developed when technology needs to be evaluated.

Moreover, there are not standard or consensual evaluation methods and tools to evaluate different types of Semantic Web technologies according to a broad range of characteristics (scalability, interoperability, usability, etc.).

1.2. The need for benchmarking in the Semantic Web

Any research advance is based on existing research results. In the case of technology, simple advances require the reuse and improvement of existing developments after they have been evaluated and compared with others. This argument, valid for any software in general, is also applicable to Semantic Web software.

The idea of benchmarking as a process that searches for improvement and best practices derives from the idea of benchmarking in the business management community [Camp, 1989, Spendolini, 1992]. This notion of benchmarking

can be found in some Software Engineering approaches [Wohlin *et al.*, 2002] but it differs from those where benchmarking is viewed as a software evaluation method for system comparison [Kitchenham, 1996, Weiss, 2002].

In this thesis, software benchmarking is defined as a collaborative and continuous process for improving software products, services, and processes by systematically evaluating and comparing them to those considered to be the best [García-Castro, 2006c].

Although software evaluation is performed inside benchmarking activities, benchmarking provides some benefits that cannot be obtained from software evaluation, such as continuous improvement of the software, recommendations for developers on the practices used when developing software, and best practices.

However, when benchmarking software, the main problem we encounter is that no software benchmarking methodology yet exists. Furthermore, the existing methodologies both for benchmarking business processes and for software evaluation and improvement in Software Engineering, such as those belonging to the Experimental Software Engineering or to the Software Measurement areas, are general methodologies and not thoroughly detailed, which makes them difficult to use in concrete cases.

The purpose in the long term is to obtain a massive improvement of the current Semantic Web technologies by providing them with reusable, consensual and freely-available evaluation methods and tools that could be used by different people in different scenarios, valid for the different types of Semantic Web tools. Therefore, a continuous evaluation and improvement of Semantic Web technologies can be possible if we perform benchmarking activities over these technologies.

This requires that

1. Generic, reusable, freely-available, and affordable methods and tools be developed for evaluating Semantic Web technologies instead of specific ones.
2. Evaluations be defined and conducted in consensus by different groups of people instead of by individual persons or organizations.
3. Evaluations be made continuously instead of being one-time activities.

1.3. Semantic Web technology interoperability

This thesis deals with one important problem of the Semantic Web, that of the interoperability of Semantic Web technologies, and also with the evaluation of this interoperability.

According to the Institute of Electrical and Electronics Engineers (IEEE), interoperability is the ability of two or more systems or components to exchange information and to use this information [IEEE-STD-610, 1991]. Duval proposes a similar definition by stating that interoperability is the ability of independently

developed software components to exchange information so they can be used together [Duval, 2004]. For us, interoperability is the ability that Semantic Web tools have to interchange ontologies and use them.

Figure 1.2 shows an example of different ontology interchanges that could occur in the Semantic Web. In this example, a user (A) develops an ontology with his favourite ontology editor and stores the ontology in a web server. Then, a remote user (B) accesses the ontology published in the Web with his own ontology editor, makes some changes in it, and uses a reasoner for evaluating the consistency of the ontology. Afterwards, the user stores the ontology in his filesystem to later use it with an annotator to annotate his personal web page using the ontology. A third remote user (C) accesses the second user's personal web page and browses its semantic information with an ontology browser.

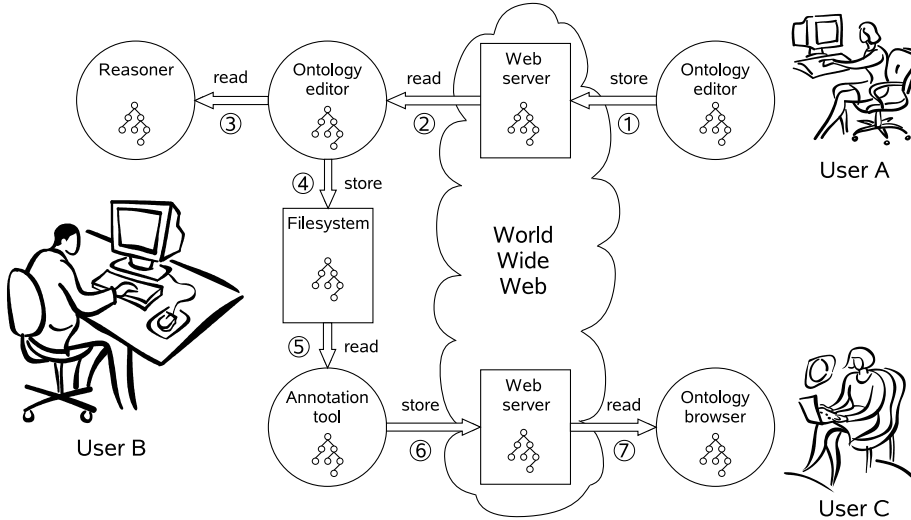


Figure 1.2: Example of ontology interchanges in the Semantic Web.

One of the factors that affects interoperability is heterogeneity. Sheth [1998] classifies the levels of heterogeneity of any information system into information heterogeneity and system heterogeneity. In this thesis, only information heterogeneity (and, therefore, interoperability) is considered, whereas system heterogeneity, which includes heterogeneity due to differences in information systems or platforms (hardware or operating systems) is disregarded.

Furthermore, interoperability is treated in this thesis in terms of knowledge reuse and must not be confused with the interoperability problem caused by the integration of resources, being the latter related to the ontology alignment problem [Euzenat *et al.*, 2004a], that is, the problem of how to find relationships between entities in different ontologies.

1.3.1. Heterogeneity in ontology representation

Ontologies enable interoperability among heterogeneous Semantic Web technologies by providing a structured, machine-processable conceptualization.

Semantic Web technologies appear in different forms (ontology development tools, ontology repositories, ontology alignment tools, reasoners, etc.) and interoperability is a must for these technologies because they need to be in communication to interchange ontologies and use them in the distributed and open environment of the Semantic Web.

On the other hand, interoperability is a problem for the Semantic Web due to the heterogeneity of the knowledge representation formalisms of the different existing systems, since each formalism provides different knowledge representation expressivity and different reasoning capabilities, as it occurs in knowledge-based systems [Brachmann and Levesque, 1985].

Current Semantic Web technologies manage different representation models, e.g., the W3C recommended languages RDF(S) and OWL, models based in Frames or in the different families of Description Logics, or other models such as the Unified Modeling Language²² (UML), the Ontology Definition Metamodel²³ (ODM), or the Open Biomedical Ontologies²⁴ (OBO) language.

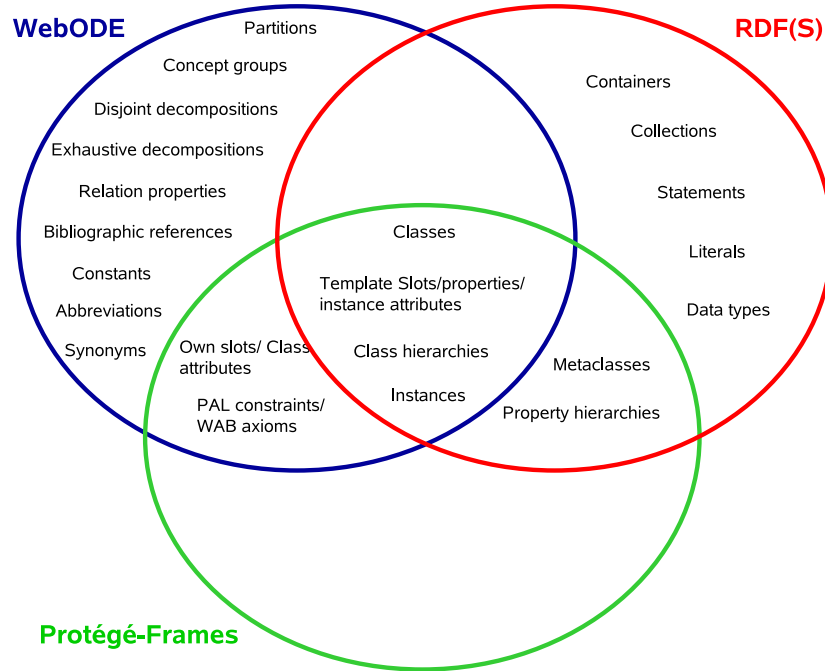


Figure 1.3: Knowledge models of Protégé-Frames, WebODE and RDF(S).

²²<http://www.uml.org/>

²³<http://www.omg.org/ontology/>

²⁴<http://obofoundry.org/>

Figure 1.3²⁵ shows an example of the heterogeneity between different representation formalisms. It provides an informal comparison of the knowledge models of two ontology editors, Protégé-Frames and WebODE (which are both frame-based), and of the RDF(S) language. The figure also indicates that some common components are included in the three knowledge models (classes, properties, class hierarchies, and instances), whereas some other components are only included in one or two of the knowledge models.

1.3.2. The interoperability problem

Figure 1.4 shows the two common ways of interchanging ontologies within Semantic Web tools: directly by storing the ontology in the destination tool, or indirectly by storing the ontology in a shared resource, such as a files server, a web server, or an ontology repository.

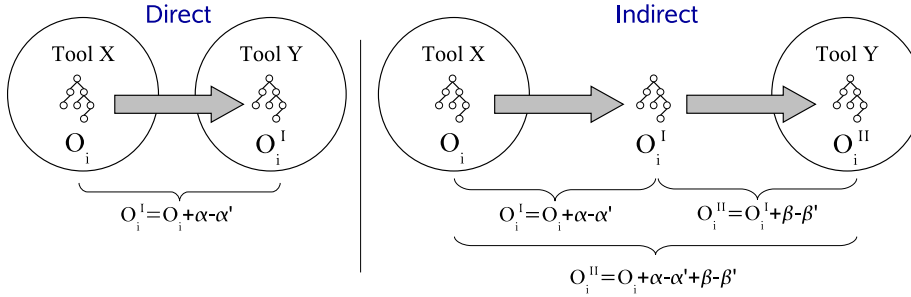


Figure 1.4: Ontology interchanges within Semantic Web tools.

The ontology interchange should pose no problems when a common representation formalism is used by all the systems involved in the interchange and there should be no differences between the original and the final ontologies (i.e., the α s and β s in the figure should be null).

However, in the real world, it is not feasible to use a single system, as each system provides different functionalities, nor it is to use a single representation formalism, since some representation formalisms are more expressive than others and different formalisms provide different reasoning capabilities, as mentioned in the previous section.

Most of the Semantic Web systems natively manage a W3C recommended language, either RDF(S), OWL, or both; but some systems manage other representation formalisms. If the systems participating in an interchange (or the shared resource) have different representation formalisms, the interchange requires at least a translation from one formalism to the another. These ontology translations from one formalism to another formalism with different expressiveness cause information additions or losses in the ontology (the α s and β s in

²⁵Inspired by the informal comparison of the Protégé and OWL knowledge models using Venn diagrams in [Knublauch, 2003b].

figure 1.4), once in the case of a direct interchange and twice in the case of an indirect one.

Due to the heterogeneity between representation formalisms in the Semantic Web scenario, the interoperability problem is highly related to the ontology translation problem that occurs when common ontologies are shared and reused over multiple representation systems [Gruber, 1993].

1.3.3. Categorising ontology differences

The differences between an ontology and the translated one can happen at different levels. Sometimes changes in one level cause changes in other levels; in other cases, changes in one level do not cause further changes in other levels.

Barrasa [2007] summarizes the different ontology heterogeneity levels according to the different classifications found in the literature [Kim and Seo, 1991, Hammer and McLeod, 1993, Visser *et al.*, 1997, Klein, 2001, Dou *et al.*, 2004, Tamma, 2001, Bouquet *et al.*, 2004, Corcho, 2005]. These levels and the classifications found in the literature can be seen in figure 1.5. The levels are

- **Lexical.** At this level we encounter all the differences related to the ability of segmenting the representation into characters and words (or symbols).
- **Syntactic.** Here we encounter all forms of heterogeneity that depend on the choice of the representation format. Some mismatches are syntactic sugar while others are caused by expressing the same thing through a totally different syntax.
- **Paradigm.** Here we encounter mismatches caused by the use of different paradigms to represent concepts such as time, action, plans, causality, etc.
- **Terminological.** At this level, we encounter all forms of mismatches related to the process of naming the entities (e.g. individuals, classes, properties, relations) that occur in an ontology.
- **Conceptual.** Here we encounter mismatches that have to do with the entities chosen to model a domain and that present differences in coverage, granularity and perspective.
- **Pragmatic.** Finally, at this level, we encounter all the discrepancies that result from the fact that different individuals/communities may interpret the same ontology in different ways in different contexts.

As mentioned above, in the current Semantic Web we can find many tools that provide specific and limited functionalities. However, not to be aware of the interoperability capabilities of the existing Semantic Web technologies causes important problems when more complex technologies and applications are built reusing existing technologies, and this ignorance regarding interoperability is mainly due to the fact that tool interoperability has not been evaluated because there is not an easy way of making this evaluation.

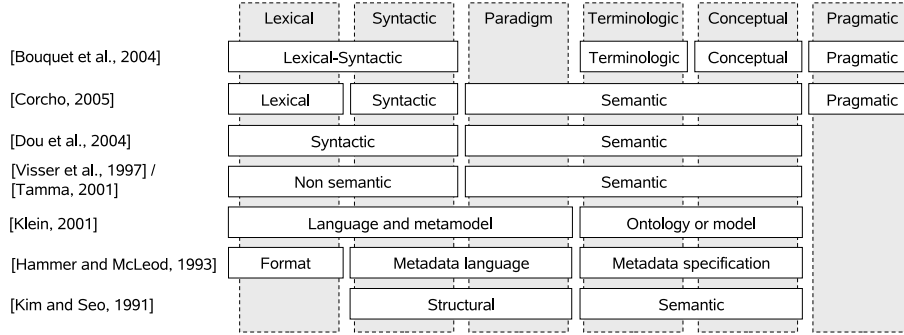


Figure 1.5: Classification of ontology heterogeneity levels [Barrasa, 2007].

As seen in previous workshops on Evaluation of Ontology-based Tools (EON) [Sure and Corcho, 2003, Corcho, 2005], the current Semantic Web tools pose problems for 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; other times, however, the problems are caused by defects in the tools. On the other hand, finding out why interoperability fails is cumbersome and non-trivial because any assumption made for translation within one tool may easily prevent successful interoperability with other tools.

1.4. Thesis contributions

I have tried to help advance research in this field by providing the contributions that follow next, which I think are quite significant:

- **Benchmarking methodology for Semantic Web technologies.** A benchmarking methodology for Semantic Web technologies has been developed as generic as possible, so it could be used in other kind of software. Furthermore, this methodology has been validated by checking that it meets the necessary and sufficient conditions that every methodology should satisfy [Paradela, 2001].

The benchmarking methodology proposed in this thesis has been used within the Knowledge Web European Network of Excellence not only in the interoperability benchmarking activities presented below but also in other benchmarking activities that involved ontology alignment tools [Euzenat *et al.*, 2004b] and reasoners [Huang *et al.*, 2007]. The general goal of all the benchmarking activities that took place in Knowledge Web was to support the industrial applicability of Semantic Web technologies.

- **The UPM Framework for Benchmarking Interoperability.** The

UPM Framework for Benchmarking Interoperability²⁶ (UPM-FBI) is publicly available and includes all the resources (experiment definitions, benchmark suites and tools) needed for benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.

The UPM-FBI includes four consensual benchmark suites that contain ontologies to be used in interoperability evaluations and two approaches for performing interoperability experiments (one manual and the other automatic), each of them containing different tools that support the execution of the experiments and the analysis of the results.

- **Interoperability benchmarking activities.** To show how the benchmarking methodology can be applied and then to validate it, such methodology has been used in two concrete case studies with the purpose of benchmarking the interoperability of Semantic Web technologies using the W3C languages. The first benchmarking (the RDF(S) Interoperability Benchmarking) contemplated interoperability using RDF(S) as the interchange language, whereas the second one (the OWL Interoperability Benchmarking) contemplated interoperability using OWL as the interchange language.

This thesis shows how these two benchmarking activities have been organized involving several international organizations; it also presents detailed interoperability results of the participating tools, which were obtained as a result of using the UPM-FBI. The results are publicly available.

1.5. Thesis structure

The rest of the thesis is structured in the following chapters:

Chapter 2 presents a survey of the current state of software evaluation and benchmarking; it also describes different evaluation and improvement methodologies, and states what benchmark suites are and how to develop them. Then, the chapter ends summarising previous interoperability evaluations that were performed over Semantic Web technologies.

Chapter 3 describes open research problems and goals, and it also describes how this thesis can contribute to this field of research with the set of assumptions, hypotheses, and restrictions taken into account.

Chapter 4 presents a methodology for benchmarking Semantic Web technologies. First, it describes the design principles contemplated when defining the methodology and the process followed to define it. Then, it details the methodology by describing its actors, process and tasks. Finally, it explains how the benchmarking methodology was applied for benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange

²⁶http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

languages, and it also explains the UPM Framework for Benchmarking Interoperability.

Chapters 5 and 6 present a detailed definition of the experiments and the benchmark suites used in the RDF(S) Interoperability Benchmarking and in the OWL Interoperability Benchmarking, respectively. They also provide the results of performing the experiments over the different tools participating in the benchmarking activities and show how the results have improved over the time.

Chapter 7 sets forth the main conclusions of this work, emphasising its main contributions. The chapter also presents future work to be performed in the fields of software benchmarking and Semantic Web interoperability benchmarking.

The first two appendixes provide further information about the RDF(S) Interoperability Benchmarking. **Appendix A** describes the method followed to identify benchmarks that cover all the possible combinations of the RDF(S) knowledge model components for the RDF(S) Interoperability Benchmarking; and **Appendix B** presents the benchmarks that compose the RDF(S) Import, Export, and Interoperability Benchmark Suites.

The next three appendixes provide further information about the OWL Interoperability Benchmarking. Thus **Appendix C** describes the method followed to identify the benchmarks that cover the combinations of the OWL Lite knowledge model components for the OWL Interoperability Benchmarking, **Appendix D** presents the benchmarks that compose the OWL Lite Import Benchmark Suite, and **Appendix E** describes the two ontologies used in the IBSE tool.

Finally, **Appendix F** provides a long summary of the thesis in Spanish.

The work presented in this thesis is mainly the result of research performed within the *Knowledge Web*²⁷ Network of Excellence (FP6-507482) and the CI-CYT project *Infraestructura tecnológica de servicios semánticos para la web semántica*²⁸ (TIN2004-02660); and has been partially supported by a FPI grant from the Spanish Ministry of Education (BES-2005-8024).

²⁷<http://knowledgeweb.semanticweb.org/>

²⁸<http://droz.dia.fi.upm.es/servicios/>

Chapter 2

State of the Art

This chapter presents a summary of the state of the art of software evaluation and benchmarking and describes one important problem encountered in the Semantic Web, the problem of the interoperability of Semantic Web technologies.

Sections 2.1 and 2.2 of this chapter provide an overview of the basic foundations of evaluation and benchmarking, respectively.

Section 2.3 includes brief descriptions of methodologies related to evaluation and improvement in the areas of business management benchmarking, Software Measurement and Experimental Software Engineering. This section does not present an exhaustive view of the different evaluation and improvement methodologies existing in these areas, since there are too many of them. The methodologies here considered are those that possess some characteristics relevant to software benchmarking, are well-known, provide detailed descriptions of their processes, and have been taken as a starting point for the work performed.

The last two sections of the chapter provide the basis for a methodological and technological support for benchmarking the interoperability of Semantic Web technologies. Section 2.4 presents what a benchmark suite is and the desirable properties that it should have and, finally, section 2.5 summarises the previous interoperability evaluations that were performed over Semantic Web technologies.

Table 2.1 shows the topics presented in this chapter and their relation with the contributions made by this thesis.

2.1. Software evaluation

Software evaluations play an important role in different areas of Software Engineering, such as Software Measurement, Experimental Software Engineering or Software Testing.

According to the ISO 14598 standard [ISO/IEC, 1999], software evaluation is *the systematic examination of the extent to which an entity is capable of fulfilling specified requirements*, considering software not just as a set of computer programs but also as the produced procedures, documentation and data.

State of the art	Thesis contribution
Software evaluation and benchmarking	Research on the foundations of software evaluation and benchmarking
Evaluation and improvement methodologies	Benchmarking methodology for Semantic Web technologies
The interoperability problem	Methodological and technological support for benchmarking interoperability
Previous interoperability evaluations	
Benchmark suites	

Table 2.1: Relationship between the state of the art and the thesis contributions.

Software evaluations can take place all along the software life cycle: they can be performed during the software development process by evaluating intermediate software products or when the development has finished.

Although evaluations are usually made inside the organization that develops the software, other groups of people who are independent of the organization, such as users or auditors, can also make them. The use of independent third parties in software evaluations can be very effective, but these evaluations are much more expensive for the organizations [Rakitin, 1997].

The goals of evaluating software depend on each specific case, but they can be summarised from [Basili *et al.*, 1986, Park *et al.*, 1996, Gediga *et al.*, 2002] as follows:

- To *describe* the software in order to understand it and to establish baselines for comparisons.
- To *assess* the software with respect to some quality requirements or criteria and determine the degree of desired quality of the software product and its weaknesses.
- To *improve* the software by finding opportunities for enhancing its quality. This improvement is measured by comparing the software with the baselines.
- To *compare* alternative software products or different versions of a same product.
- To *control* the software quality by ensuring that it meets the required level of quality.
- To *foresee* in order to take decisions, establishing new goals and plans for accomplishing them.

Software can be evaluated according to numerous quality attributes. Multiple **software quality models** have been defined after the first proposals made by Boehm [1976] and Calvano and McCall [1978] in the 1970's. In this thesis,

these quality models are not described in detail, and only an example of the models is provided, illustrated with one of the most well-known frameworks for software product quality, the framework described in the ISO 9126 standard [ISO/IEC, 2001].

The ISO 9126 identifies three different views of software product quality:

- *Internal quality.* Internal quality concerns the totality of the characteristics of the software product from an internal view. Details of software product quality can be improved during the implementation, review and test of the code; however, the fundamental nature of the software product quality represented by internal quality remains unchanged unless re-designed.
- *External quality.* External quality concerns the totality of the characteristics of the software product from an external view and refers to the quality of the software when this is executed; quality is typically measured and evaluated while testing the software in a simulated environment with simulated data using external metrics. During testing, most faults should be discovered and eliminated, but some faults may still remain afterwards. However, because it is difficult to correct the software architecture or other basic design aspects of the software, the fundamental design usually remains unchanged throughout testing.
- *Quality in use.* Quality in use refers to the user's view of the software product quality when this is used in a specific environment and in a specific context. It measures the extent to which users can achieve their goals in a particular environment, rather than the properties of the software itself.

The quality model for internal and external quality proposes six high-level software quality characteristics, which are decomposed into sets of subcharacteristics. These high-level characteristics, shown in figure 2.1, are the following:

- *Functionality.* It is the capability of the software to provide functions that meet stated and implied needs when the software is used under specified conditions. Functionality can be decomposed into suitability, accuracy, interoperability, security, and functionality compliance.
- *Reliability.* It is the capability of the software to maintain its level of performance when used under specified conditions. Reliability can be decomposed into maturity, fault tolerance, recoverability, and reliability compliance.
- *Usability.* It is the capability of the software to be attractive and understood, learned, and used by the user, when it is employed under specified conditions. Usability can be decomposed into understandability, learnability, operability, attractiveness, and usability compliance.

- *Efficiency.* It is the capability of the software to provide appropriate performance, relative to the amount of resources used, under stated conditions. Efficiency can be decomposed into time behaviour, resource utilisation, and efficiency compliance.
- *Maintainability.* It is the capability of the software to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment and in requirements and functional specifications. Maintainability can be decomposed into analysability, changeability, stability, testability, and maintainability compliance.
- *Portability.* It is the capability of software to be transferred from one environment to another. Portability can be decomposed into adaptability, installability, co-existence, replaceability, and portability compliance.

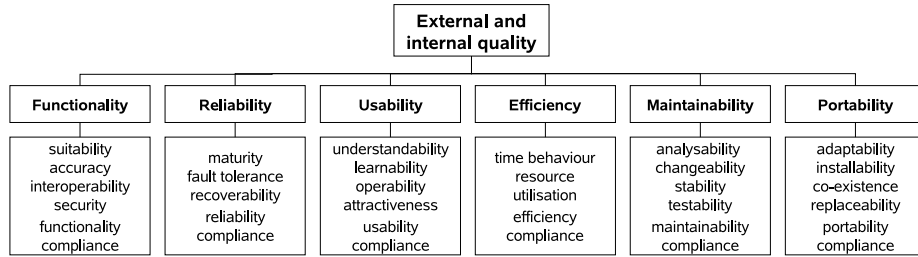


Figure 2.1: Quality model for internal and external quality [ISO/IEC, 2001].

The quality model for quality in use proposes four software quality characteristics, shown in figure 2.2. These characteristics are the following:

- *Effectiveness.* It is the capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use.
- *Productivity.* It is the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use.
- *Safety.* It is the capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.
- *Satisfaction.* It is the capability of the software product to satisfy users in a specified context of use.

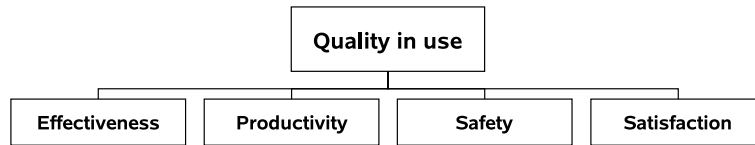


Figure 2.2: Quality model for quality in use [ISO/IEC, 2001].

2.2. Benchmarking

In the last decades, the word benchmarking has become relevant within the business management community. The most well-known definitions in this area are those of Camp [1989] and Spendolini [1992]. Camp defines benchmarking as *the search for industry best practices that lead to superior performance*, while Spendolini expands Camp’s definition by adding that benchmarking is *a continuous, systematic process for evaluating the products, services, and work processes of organizations that are recognised as representing best practices for the purpose of organizational improvement*. In this context, best practices are good practices that have worked well elsewhere, are proven and have produced successful results [Wireman, 2003]. These definitions highlight the two main benchmarking characteristics: continuous improvement and the search for best practices.

The Software Engineering community also uses the term benchmarking though it does not share a common benchmarking definition. Below some of the most representative definitions used by the Software Engineering community are presented:

- Kitchenham [1996] and Weiss [2002] define benchmarking as a software evaluation method suitable for system comparisons. For Kitchenham, benchmarking is *the process of running a number of standard tests using a number of alternative tools/methods and assessing the relative performance of the tools in those tests*, whereas for Weiss, benchmarking is *a method of measuring performance against a standard or a given set of standards*.
- Wohlin et al. [2002] adopt the business benchmarking definition, viewing benchmarking as a continuous improvement process that strives to be the best of the best through the comparison of similar processes in different contexts.

2.2.1. Benchmarking vs evaluation

The reason for benchmarking software products instead of just evaluating them is to obtain several benefits that cannot be obtained from software evaluations. As figure 2.3 illustrates, software evaluation shows the weaknesses of the

software or its compliance to quality requirements. If several software products are involved in the evaluation, we also obtain a comparative analysis of these products and recommendations for users. However, when benchmarking several software products, in addition to all the benefits commented, we also gain continuous improvement of the products, recommendations for developers on the practices used when developing these products and, from these practices, those that can be considered best practices.

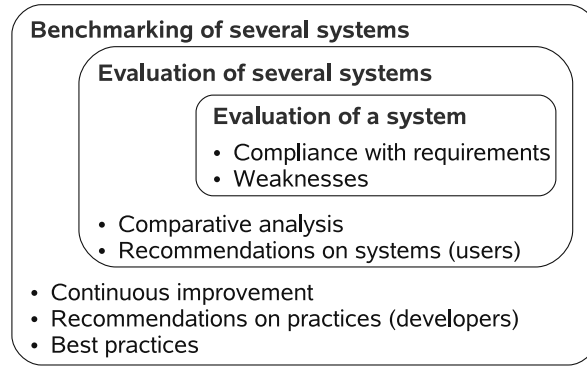


Figure 2.3: Benchmarking benefits.

2.2.2. Benchmarking classifications

This section presents two different classifications of benchmarking that, although they were created inside the business management community, can be applied to software benchmarking. One of the classifications is focused on the participants involved in it, whereas the other is based on the nature of the objects under analysis.

The main benchmarking classification was presented by Camp [1989]. He categorises benchmarking depending on the kind of participants involved, and his classification has been adopted by authors such as Sole and Bist [1995], Ahmed and Rafiq [1998] and Fernandez et al. [2001]. The four categories identified by Camp are

- **Internal benchmarking.** It measures and compares the performance of activities, functions and processes within one organization.
- **Competitive benchmarking.** In this case, the comparison is made with products, services, and/or business processes of a direct competitor.
- **Functional benchmarking** (also called industry benchmarking). This category is similar to the previous one, competitive benchmarking, except that the comparison involves a larger and more broadly defined group of competitors in the same industry.

- **Generic benchmarking.** Its aim is to search for general best practices, without regarding any specific industry.

Another classification categorises benchmarking according to the nature of the objects under analysis. This classification first appeared in Ahmed and Rafiq's [1998] paper and complements Camp's classification. A few years later, Lankford [2000] established a separate classification and identified the following types of benchmarking:

- **Process benchmarking.** It involves comparisons between discrete work processes and systems.
- **Performance benchmarking.** It involves comparison and scrutiny of performance attributes of products and services.
- **Strategic benchmarking.** It involves comparison of the strategic issues or processes of an organization.

2.3. Evaluation and improvement methodologies

This section includes brief descriptions of methodologies related to evaluation and improvement in the areas of business management benchmarking, Software Measurement and Experimental Software Engineering. These three areas provide an overview of different methods that deal with issues relevant to software benchmarking, such as evaluation as a continuous activity, a company-wide perspective of software evaluation, and software experimentation, respectively.

The first part of this section presents benchmarking methodologies from the *business management* area. These methodologies view benchmarking as a mechanism to improve business processes and also consider that benchmarking is a continuous process; such methodologies include ideas that can be easily adapted to the Software Engineering community. The second part of this section presents *Software Measurement* methodologies, which are used to implement software metric programs in companies. Such methodologies are included here because they deal with the implications of software measurement at a company-wide level and because they also view software measurement as a continuous process. Finally, the third part of this section presents *Experimental Software Engineering* methodologies, which are used to perform experiments over software. Unlike in the previous approaches, software experimentation is viewed as a one-time activity. Nevertheless, these methodologies are relevant to software benchmarking because experiments over software take place inside benchmarking activities.

This list of methodologies is neither exhaustive nor complete, since methodologies from other areas such as Software Evaluation [ISO/IEC, 1999, Basili, 1985] or Software Testing [Myers *et al.*, 2004], which have not been dealt with here, could also be considered.

2.3.1. Benchmarking methodologies

This section presents the most relevant benchmarking methodologies from the business management community. The first methodology was proposed by Robert Camp [1989], who initiated the Xerox's benchmarking program and is generally regarded as the guru of the benchmarking movement. On the other hand, Xerox was the pioneer in American companies in the practice of benchmarking. The second methodology is the benchmarking wheel designed by Andersen and Pettersen [1996] after analysing about sixty different benchmarking processes in companies. Finally, the third methodology is the one proposed by the American Productivity and Quality Centre [Gee *et al.*, 2001], which is one of the premier benchmarking methodologies in the world.

These methodologies view benchmarking as a mechanism to improve business processes. All of them have similar elements, consider that benchmarking is mainly performed by a single company, and coincide in that benchmarking is a continuous process. Therefore, the steps proposed are just an iteration of the benchmarking cycle. As these methodologies are quite general, they can be easily adapted for software benchmarking.

The methodology proposed by **Camp** [1989] includes the following four phases:

- *Planning* phase. Its objective is to schedule the benchmarking investigations. The essential steps of this phase are
 - To identify what is to be benchmarked.
 - To identify comparative companies.
 - To determine the data collection method and collect data.
- *Analysis* phase. This phase involves a careful understanding of the current process practices as well as of those practices of benchmarking partners. The steps to follow in this phase are
 - To determine the current performance gap between practices.
 - To project the future performance levels.
- *Integration* phase. This phase involves planning to incorporate the new practices obtained from benchmark findings in the organization. The main step of this phase is
 - To communicate benchmark findings and to gain acceptance.
- *Action* phase. In this phase, benchmarking findings and operational principles based on the findings are converted into actions. The steps recommended are
 - To establish functional goals.
 - To develop action plans.

- To implement specific actions and monitor progress.
- To recalibrate benchmarks.

Camp also identifies a *maturity state* that will be reached when best industry practices are incorporated into all business processes and benchmarking becomes institutionalised.

The benchmarking process proposed by **Andersen and Pettersen** [1996] is called the benchmarking wheel. Such process is composed of five phases that have a degree of overlap:

- *Plan* phase. This is the most important phase since thorough planning is key to create the foundation for an effective benchmarking that produces good results. The steps to follow in this phase are
 - To select the process to be benchmarked, based on the company's strategy.
 - To form the benchmarking team.
 - To understand and document the process.
 - To establish performance measures for the process (quality, time, and cost).
- *Search* phase. This phase is devoted to searching and selecting suitable benchmarking partners. The steps to follow in this phase are
 - To design a list of criteria that an ideal benchmarking partner should satisfy.
 - To search for potential benchmarking partners, i.e., who performs the process in question better than the company.
 - To compare the candidates and select benchmarking partner(s).
 - To establish contact with the selected benchmarking partner(s) and gain acceptance for participating in the benchmarking study.
- *Observe* phase. The goal of this phase is to observe the benchmarking partners and obtain their performance levels, the practices or methods that make it possible to achieve these levels, and the enablers that permit performing the process according to these practices or methods. The steps to follow in this phase are
 - To assess information needs and sources.
 - To select methods and tools for collecting information and data.
 - To observe and debrief.
- *Analyse* phase. The main purpose of this phase is to find out gaps in performance levels between own and partners' process, the root causes of the gaps, and the enablers that particularly contribute to the gaps. The steps to follow in this phase are

- To sort the collected information and data.
 - To quality control and normalise the information and data.
 - To identify gaps in performance levels.
 - To identify the causes for the gaps.
- *Adapt* phase. In this phase, the findings from the *Analysis* phase must be adapted to the organization's own conditions and implemented in the company. The steps to follow in this phase are
- To communicate the findings from the analysis and gain acceptance through participation and information.
 - To establish functional goals for the improvements that match the other improvement plans of the company.
 - To design an implementation plan for the improvements.
 - To put the implementation plan into action.
 - To monitor the progress and adjust deviations.
 - To close the benchmarking study with a final report.

Another methodology is the one proposed by the **American Productivity and Quality Centre**. This methodology has been broken down by Gee et al. [2001] into the following four phases:

- *Plan* phase. Its goal is to prepare the benchmarking study plan, select the team and partners, and analyse the organizational process. The steps to follow are
- To form (and train, if needed) the benchmarking team.
 - To analyse and document the current process.
 - To identify the area of study on which the team will focus.
 - To identify the most important customer.
 - To identify the smaller subprocesses, especially problem areas.
 - To identify the critical success factors for the area and develop measures for them.
 - To establish the scope of the benchmarking study.
 - To develop a purpose statement.
 - To develop criteria for determining and evaluating prospective benchmarking partners.
 - To identify target benchmarking partners.
 - To define a data collection plan and determine how the data will be used, managed, and distributed.
 - To identify how implementation of improvements will be accomplished.

- *Collect* phase. The goals of the data collection phase are to prepare and administer questions, capture the results, and follow-up partners. The steps to follow here are
 - To collect secondary benchmarking information in order to determine whom to target as benchmarking partners.
 - To collect primary benchmarking data from the benchmarking partners.
- *Analyse* phase. The goals of this phase are to analyse performance gaps and identify best practices, methods, and enablers. The steps to follow here are
 - To compare one's current performance data with the partner's data.
 - To identify any operational best practices observed and the factors and practices that facilitate superior performance.
 - To formulate a strategy to close any identified gaps.
 - To develop an implementation plan.
- *Adapt* phase. The goals of this phase are to publish findings, create an improvement plan, and execute the plan. The steps to follow here are
 - To implement the plan.
 - To monitor and report progress.
 - To document and communicate the study results.
 - To plan for continuous improvement.

Gee et al. [2001] also identify the *final steps* that should be carried out after the *Adapt* phase. These steps are

- To document the benchmarking in a final report and capture any lessons learned that can be of future value, capturing also a variety of process information.
- To communicate the results of the benchmarking effort to the management and staff.
- To send a copy of the final report to the benchmarking partners.
- To routinely review the performance of the benchmarked processes and ensure that goals are being met.
- To move on to what is next by identifying other candidate processes for benchmarking.

It can be observed that the previous methodologies contain some similar tasks. Table 2.2 shows a comparison of the tasks that each of these methodologies deal with, whereas table 2.3 identifies the set of common tasks that these methodologies treat. These common tasks have been grouped into phases according to the phases used in the methodologies.

[Camp, 1989]	[Andersen and Petersen, 1996]	[Gee et al., 2001]
Identify what is to be benchmarked	Select the process to be benchmarked	
	Form the benchmarking team	Form (and train, if needed) benchmarking team
	Understand and document the process	Analyse and document the current process
		Establish scope of benchmarking study
		Develop purpose statement
		Identify target benchmarking partners
		Define data collection plan and methods
		Identify improvements implementation methods
	Establish process performance measures	
	Design benchmarking partner criteria	Develop criteria for benchmarking partners
	Search for potential benchmarking partners	Secondary research based on select/sort criteria
		Evaluate results and ...
Identify comparative partners	Compare and select benchmarking partners	... identify potential partners
		Develop data collection instruments
		Pilot data collection instruments internally
		Review the secondary information
	Contact partners and gain participation	Identify, contact and enlist best practice partners
		Screen partners and evaluate for best “fit”
	Assess information needs and sources	
Determine data collection method and ...	Select method and tool for data collection	Develop detailed questionnaire
... collect data	Observe and debrief	Conduct detailed investigation
	Sort the collected data	
	Quality control the data	
Determine current performance gap	Identify gaps in performance levels	Compare your data to your partners’ data
	Identify the causes for the gaps	Identify operational best practices and enablers
Project future performance levels		
Communicate findings and gain acceptance	Communicate findings and gain acceptance	
Establish functional goals	Establish functional goals	Formulate strategy to close the gaps
Develop action plans	Design an implementation plan	Develop implementation plan
Implement actions and monitor progress	Put the implementation plan into action	Implement the plan
	Monitor progress and adjust deviations	Monitor and report progress
	Write a final report	Document the study
Recalibrate benchmarks		
		Plan for continuous improvement

Table 2.2: Comparison of tasks in benchmarking methodologies.

Phase	Task
Plan	Identify the process to be benchmarked
	Form and train the benchmarking team
	Analyse and document the current process
Data collection	Define criteria for benchmarking partners
	Identify potential partners
	Contact the potential partners and enlist
	Define method and tools for data collection
	Collect data
Analysis	Determine the current performance gaps
	Identify the causes for the gaps
Change/adapt	Communicate findings and gain acceptance
	Formulate strategy to close the gaps
	Develop the implementation plan
	Implement the plan
	Monitor and report progress
	Document the study

Table 2.3: Common tasks in benchmarking methodologies.

2.3.2. Software Measurement methodologies

This section describes the most relevant Software Measurement methodologies. First, it presents the methodology proposed by Grady and Caswell [1987], which was extracted from the first and most extensive implementation of a company-wide software metrics program in Hewlett-Packard. Then, it expounds the methodologies of Goodman [1993] and McAndrews [1993], which are based on other software measurement methodologies.

These methodologies have similar elements and are used to implement software metric programs in companies. They regard the measurement activity as an activity mainly performed by a single company, and coincide in the fact that such activity is a continuous process.

Grady and Caswell [1987] describe the implementation of a software metrics program in Hewlett-Packard. They propose the following steps to define and implement metrics in an organization:

- *To define company/project objectives for the program.* The objectives you define will frame the methods you use, the costs you are willing to incur, the urgency of the program, and the level of support you have from your managers.
- *To assign responsibilities.* The organizational location of responsibilities for software metrics and the specific people you recruit to implement your objectives is a signal to the rest of the organization that indicates the importance of the software metrics program.

- *To do research.* Examining data external to the organization in order to get ideas for conducting experiments and set expectations for the results.
- *To define initial metrics to be collected.* You can start with a simple set.
- *To sell the initial collection of these metrics.* The success of a metrics program depends on the accuracy of the data collected, and this accuracy relies on the commitment of the personnel involved and the time required to collect them.
- *To get tools for automatic data collection and analysis.* Such type of tools helps simplify the task of collection, reduce the time expenditure, ensure accuracy and consistency, and reduce psychological barriers to collection.
- *To establish a training class in software metrics.* Training classes help ensure that the objectives for data collection are framed in the context of the company/project objectives. Training is also necessary to achieve the widespread usage of metrics.
- *To publicize success stories and encourage the exchange of ideas.* Making public success stories provides feedback to the people taking measurements that confirms that their work is valuable. It also helps spread these successes to other parts of the organization.
- *To create a metrics database.* A database of collected measurements is necessary to evaluate overall organizational trends and effectiveness. Such database also provides valuable feedback concerning whether the metric definitions used are adequate.
- *To establish a mechanism for changing the standard in an orderly way.* As the organization now understands its development process better, the process and the metrics collected will evolve and mature. For that, there must be a mechanism in place that basically repeats the previous steps.

Grady and Caswell also state that a software metrics program must not have a strategy into itself. Collecting software metrics must not be an isolated goal but a part of an overall strategy for improvement.

Goodman [1993] sets up a framework for developing and implementing software metrics programs within organizations. He defines a generic model tailored to each specific environment. The stages proposed for the model are the following:

- *Initialisation stage.* This stage is caused by some trigger, and driven by an initiator. In this stage the initial scope of the program is defined.
- *Requirements definition.* This stage involves finding out what the various parts of the organization want from a software metrics program. Such stage is concerned with requirements gathering and specification.

- *Component design.* This stage encompasses both the choice of specific metrics and the design of the infrastructure that will support the use of those metrics.
- *Component build.* This stage involves building the components of the software metrics program regarding the requirements and design obtained in the previous stages.
- *Implementation.* This stage involves implementing the components that form the measurement initiative into the organization.

McAndrews [1993] proposes a method for establishing a measurement process as part of an organization's overall software process. The phases proposed for the process are the following:

- *To identify scope.* The measurement process is originated by the need for measurement. It is important to understand this need and the audiences of the process before designing such a process. This phase helps establish the purpose of measurement by identifying the objectives that such measurement is to support, the issues involved to meet these objectives, and the measures that provide insight into these issues. The tasks that this phase comprises are related to the identification of
 - The needs for measurement.
 - The organization's objectives to be addressed by each measurement need.
 - The methods to be used to achieve the objectives.
 - The issues that need to be managed, controlled, or observed.
 - The precise, quantifiable, and unambiguous measurement goals in which each measurement issue is translated into.
 - A list of questions for each goal that, when answered, will determine if the goal is being achieved.
 - A list of questions for each goal that, when answered, will determine if the goal is being achieved.
- *To define procedures.* Once measures have been identified, operational definitions and procedures of the measurement process should be constructed and documented. In this phase, the measures and counting procedures must be defined; in addition, forms or procedures to record the measures must be constructed along with a database or spreadsheet to store the information, measurement report formats, and mechanisms to provide feedback; then analysis methods must be defined, and potential relationships among the measures must be identified.
- *To collect data.* Here data are collected, recorded, and stored, using the operational definition of the measurement procedures. Then, data are validated and the procedures are reviewed for adequacy.

- *To analyse data.* This phase consists in analysing the data required for preparing the reports, presenting the reports to the audience, and reviewing procedures for accuracy and adequacy.
- *To evolve the process.* This phase is used to improve the process and ensure a structured way of incorporating issues and concerns during this process. This phase also assesses the adequacy of the measurement process itself.

As can be observed, the previous methodologies contain some similar tasks. Table 2.4 identifies the set of common tasks that these methodologies treat; these common tasks have been grouped into phases according to the phases used in the methodologies. Table 2.5 shows a comparison of the tasks dealt with in each of these methodologies.

Phase	Task
Plan	Identify the needs for measurement
	Define organization objectives
	Assign responsibilities
	Research external products
Define	Define metrics to collect
	Define data collection methods
	Define storage/analysis/feedback methods
	Establish training mechanisms
	Build the metrics database
Implement	Collect data
	Analyse data
	Change
	Disseminate results

Table 2.4: Common tasks in Software Measurement methodologies.

2.3.3. Experimental Software Engineering methodologies

This section presents the most relevant Experimental Software Engineering methodologies. Such methodologies are pertinent to software benchmarking because experiments over software take place inside benchmarking activities. Then, the approaches of the main authors in this field are presented: from the vision of Basili, who shaped the field of Experimental Software Engineering from its very start, to the outstanding software experimentation process proposed by Wohlin et al.

These methodologies are used to perform software experiments; they have similar elements, and treat software experimentation as a one-time activity. On the other hand, some of these methodologies allow the participation of several teams in the experimentation even though the experiments are usually intended to be performed by one software development team.

[Grady and Caswell, 1987]	[Goodman, 1993]	[McAndrews, 1993]
Define the objectives for the program	Initial management decision	Identify the needs for measurement
Assign responsibilities	Assign management responsibility	Identify the relevant organization objectives
	Perform feasibility study	
	Customer and market identification	
	Senior management presentation	
	Initial publicity	
	Identify available data sources	
	Identify storage and feedback requirements	
	Identify external products	
Do research		Identify the methods to achieve the objectives
		Identify the issues to manage/control/observe
Define initial metrics to collect	Basic metric specification	Define the measures
Sell the initial collection of these metrics	Map base metrics to available data	
Get tools for data collection and analysis	Define and build data collection mechanisms	Define counting procedures
		Define procedures to record the measures
		Design a database to store the information
	Define storage/analysis/feedback mechanisms	Define analysis and feedback mechanisms
	Prepare business and marketing plan	
	Define infrastructure	
Establish a training class	Define support training	
	Select implementation team	
	Document techniques	
	Build components and training material	
Create a metrics database	Build metrics database	
	Launch planning and prelaunch publicity	
	Launch the program	Collect data
		Record and store data
		Review and revise procedures
		Analyse data and prepare report
Publicise success stories		Present report
Establish a mechanism for change	Implementation	Review and revise procedures
		Evaluate progress and the process

Table 2.5: Comparison of tasks in Software Measurement methodologies.

Basili et al. [1986] propose a framework for conducting experimentation that includes the following four phases:

- *Definition.* Its goal is to identify the motivation, object, purpose, perspective, domain, and scope of the experiments.
- *Planning.* Its goal is to design the experiments, choose the criteria to be used according to the experiment definitions, and define the measurement process.
- *Operation.* This phase consists in preparing and executing the experiments and then analysing the data obtained after their execution.
- *Interpretation.* Here, the results of the previous phase are interpreted in different contexts, extrapolated to other environments, and then presented; finally, the modifications needed are made.

Pfleeger [1995] proposes the following six steps for carrying out a software experiment:

- *Conception.* Here the goals of the experiment must be clearly and precisely stated so that the experiment to be conducted will provide the answers to these goals. This step also includes analysing whether a formal experiment is the most appropriate research technique to use.
- *Design.* Here the objective is translated into a formal hypothesis. To test such hypothesis, a formal experiment design is generated. This experiment design describes how the tests that the experiment includes will be organized and run; it also defines the following issues:
 - Characteristics of the people who will perform the experiment.
 - A baseline of information to make comparisons.
 - Criteria for measuring and judging effects.
 - Methods for obtaining the measures.
- *Preparation.* This step involves readying the subjects for the experiment. To do this, instructions must be properly stated and, if possible, a dry run of the experiment should be carried out.
- *Execution.* Here the experiment is executed according to the experiment plan.
- *Analysis.* This step can be divided into two substeps: in the first one, all the measurements must be reviewed to make sure that they are valid and useful; in the second, the measurements must be analysed using statistical principles. This analysis will support or refuse the hypothesis previously stated.

- *Dissemination and decision-making.* Here the experiment conclusions must be documented to allow duplicating the experiment and confirming these conclusions in similar settings. The experimental results may be used for three different purposes:

- To support decisions about how to develop or maintain software in the future.
- To suggest changes to others' development environments.
- To perform similar experiments with variations in order to understand how the results are affected by carefully controlled changes.

Wohlin et al. [2000] propose a general experiment process that includes the following phases:

- *Definition.* Here the goals of the experiment are defined by identifying
 - Its object, which can be products, processes, resources, models, metrics or theories.
 - Its purpose, or the intention of the experiment.
 - Its quality focus, or the effect to be studied.
 - Its perspective, or the viewpoint from which the experiment results are interpreted.
 - Its context, or the environment where the experiment is run.
- *Planning.* In this phase it is defined how the experiment is to be conducted by performing the following tasks: context selection, to characterize the experiment context; hypothesis formulation, to formalise the experiment into hypotheses; variables selection, to select the independent and dependent variables; selection of subjects, to select the subjects from the desired population; experiment design, to describe how the tests of the experiment are organized and run; instrumentation, to develop instruments to perform the experiment and to monitor it; and validity evaluation, to ensure the validity of the experiment results.
- *Operation.* This phase involves carrying out the experiment and collecting the data to be analysed by performing the following tasks: preparation, to prepare the people and materials needed to perform the experiments; execution, to execute the experiment and collect data; and data validation, to check that the data is reasonable and that it has been collected correctly.
- *Analysis and interpretation.* In this phase, descriptive statistics are used to understand the collected data, considering a possible reduction of the data set. After the data has been reduced, a hypothesis test is performed using statistical techniques.

- *Presentation and package.* This phase deals with the packaging of the experiment findings in an experiment report and with the presentation of these findings. For replicating the experiment, it is necessary to take into account the information required.

Kitchenham et al. [2002] do not propose a methodology but a set of guidelines for carrying out experiments. These guidelines explain what to do and not to do in the following six basic experimentation areas:

- *Experimental context*
 - Specify as much as possible the industrial context. Define clearly the entities, attributes, and measures capturing the contextual information.
 - If a specific hypothesis is being tested, state it clearly prior to performing the tests and discuss the theory from which it is derived, so that its implications are apparent.
 - If the research is exploratory, state clearly and, prior to the data analysis, what questions the investigation is intended to address, and how it will address them.
 - Describe research that is similar to the present research and how current work relates to it.
- *Experimental design*
 - Identify the population from which the experimental subjects and objects are drawn.
 - Define the process by which the subjects and objects are selected and assigned to treatments.
 - Restrict yourself to simple study designs or, at least, designs that are fully analysed in the literature.
 - Define the experimental unit.
 - For formal experiments, perform a pre-experiment or pre-calculation to identify or estimate the minimum required sample size.
 - Use appropriate levels of blinding.
 - Make explicit any vested interests, and report what has been done to minimise bias.
 - Avoid the use of controls unless you are sure that the control situation can be unambiguously defined.
 - Fully define all interventions.
 - Justify the choice of outcome measures in terms of their relevance to the objectives of the empirical study.
- *Conducting the experiment and data collection*

- Define all software measures fully, including the entity, attribute, unit, and counting rules.
 - For subjective measures, present a measure of inter-rater agreement.
 - Describe any quality control method used to ensure completeness and accuracy of data collection.
 - For surveys, monitor and report the response rate, and discuss the representativeness of the responses and the impact of non-response.
 - For observational studies and experiments, record data about subjects who drop out from the studies. Also record other performance measures that may be adversely affected by the treatment, even if they are not the main focus of the study.
- *Analysis*
 - Specify all the procedures used to control multiple testing.
 - Consider using blind analysis.
 - Perform sensitivity analysis.
 - Ensure that the data do not violate the assumptions of the tests used on them.
 - Apply appropriate quality control procedures to verify your results.
 - *Presentation of results*
 - Describe or cite a reference for all statistical procedures used.
 - Report the statistical package used.
 - Present quantitative results showing the magnitude of effects and the confidence limits.
 - Present the raw data whenever possible. Otherwise, confirm that they are available for confidential review.
 - Provide appropriate descriptive statistics and graphics.
 - *Interpretation of results*
 - Define the population to which inferential statistics and predictive models apply.
 - Define the type of study taken into account.
 - Differentiate between statistical significance and practical importance.
 - Specify any limitations of the study.

As can be observed, the previous methodologies contain some similar tasks. Table 2.6 shows a comparison of the tasks dealt with in each of these methodologies, whereas table 2.7 identifies the set of common tasks that these methodologies tackle. These common tasks have been grouped into phases according to the phases used in the methodologies.

[Basili et al., 1986]	[Pfleeger, 1995]	[Wohlin et al, 2000]	[Kitchenham et al., 2002]
Define the motivation			
Define the object		Object of the study	
Define the purpose	State the objective of the study	Purpose	State questions intended to address
		Quality focus	
Define the perspective		Perspective	
Define the domain		Context	Industrial context
Define the scope			
		Context selection	
	Translate the objective into a hypothesis	Hypothesis formulation	State hypothesis
		Variables selection	
		Selection of subjects	Define entities, attributes and measures
			Describe similar research
Design the experiment	Design the experiment	Experiment design	
	Identify the participants		Identify the population
			Define the experimental unit
		Instrumentation	
	Establish a baseline for comparison		
		Validity evaluation	
Define the criteria	Define criteria for measuring and judging effects		Define software measures
Define the metrics			Define the sampling process
Preparation - pilot study	Define the method for obtaining the measures		
Data collection and ...	Prepare the subjects for the experiment	Preparation	
	Execute the experiment	Execution	
			Define quality control method
... validation	Review all the measurements taken	Data validation	Verify results
Analysis	Analyse data according to statistical principles	Descriptive statistics	Perform analysis
		Data set reduction	
		Hypothesis testing	
Interpretation context			
Extrapolation			
Impact			
	Document the experiment and the conclusions	Experiment report	
		Present findings	Present results
	Make decisions from the results		Interpret results

Table 2.6: Comparison of tasks in Experimental Software Engineering methodologies.

Phase	Task
Definition	Define the subject
	Define the purpose
	Define the perspective
	Define the context
Design	Define hypothesis
	Design the experiment
	Define the criteria
	Define the metrics
Execution	Define the method for obtaining the measures
	Prepare the subject for the experiment
	Execute the experiment
	Validate the data obtained
Analysis	Analyse the data collected
	Document the experiment
	Interpret results
Dissemination	Present results

Table 2.7: Common tasks in Experimental Software Engineering methodologies.

2.4. Benchmark suites

A benchmark suite is a collection of benchmarks, being a benchmark *a test or set of tests used for comparing the performance of alternative tools or techniques* [Sim *et al.*, 2003].

From the previous definition, we can infer that the idea of benchmark is highly related to the notion of test, the former having software comparison as a goal. Because of this, benchmark suite is sometimes used as a synonym of test suite (a group of related tests that are usually run together [Burnstein, 2003]).

Furthermore, a benchmark is specified in the same way as a test case, and we should remember that a test case specification has the following structure [IEEE, 1998]:

- *Test case specification identifier.* The unique identifier assigned to the test case specification.
- *Test items.* The items and features to be exercised by the test case.
- *Input specifications.* Each input required to execute the test case, including all the required relationships between inputs (e.g., timing).
- *Output specifications.* All of the outputs and features (e.g., response time) required of the test items.

- *Environmental needs.* The characteristics and configurations of the hardware required to execute the test case, the system and application software required to execute the test case, and any other requirements such as unique facility needs or specially trained personnel.
- *Special procedural requirements.* Any special constraints on the test procedures that execute the test case. These constraints may involve special set up, operator intervention, output determination procedures, and special wrap up.
- *Intercase dependencies.* The identifiers of test cases that must be executed prior to the test case, including the nature of the dependencies.

The following properties, extracted from different authors [Bull *et al.*, 1999, Shirazi *et al.*, 1999, Sim *et al.*, 2003, Stefani *et al.*, 2003], can help both to develop new benchmark suites and to assess the quality of different benchmark suites before using them.

Although a good benchmark suite should have most of these properties, each evaluation will require that some of them be considered before others.

It must also be reckoned that achieving a high degree of all these properties in a benchmark suite is not possible since the increment of some properties has a negative influence on others.

- **Accessibility.** A benchmark suite must be accessible to anyone interested. This involves providing the necessary software to execute the benchmark suite, and providing its documentation and its source code in order to increase transparency.

The results obtained when executing the benchmark suite should be made public so that anyone can apply the benchmark suite and compare his/her results with the results available.

- **Affordability.** Using a benchmark suite entails a number of costs, commonly in human, software, and hardware resources. The costs of using a benchmark suite must be lower than the costs of defining, implementing, and carrying out any other experiments that fulfil the same goal.

On the other hand, reducing the resources consumed in the execution of a benchmark suite can be achieved as follows: by automating the execution of the benchmark suite, by providing components for data collection and analysis, or by facilitating its use to different heterogeneous systems.

- **Simplicity.** The benchmark suite must be simple and interpretable. It must be well documented, so that anyone wanting to use it must be able to understand how it works and the results that it yields. If the benchmark suite is not transparent enough, its results will be questioned and the benchmark could be interpreted incorrectly.

To ease the process, the elements of the benchmark suite should have a common structure, use, inputs, and outputs. Measurements should have the same meanings across the benchmark suite.

- **Representativity.** The actions that perform the benchmarks composing the benchmark suite must be representative of the actions usually performed on the system.
- **Portability.** The benchmark suite should be executed on a range as wide as possible of environments, and should be applicable to as many systems as possible.

The benchmark suite should also be specified at a high enough level of abstraction to ensure that it is portable to different tools and techniques and that it is not biased against other technologies.

- **Scalability.** The benchmark suite should be parameterised to allow scaling the benchmarks with varying input rates.

It should also scale well to working with tools or techniques at different levels of maturity. It should be applicable to research prototypes and commercial products.

- **Robustness.** The benchmark suite must contemplate unpredictable environment behaviours and should not be sensitive to factors irrelevant to the study. When running the same benchmark suite on a given system under the same conditions several times, the results obtained should not change considerably.
- **Consensus.** The benchmark suite must be developed by experts so that they apply their knowledge of the domain and identify the key problems. The benchmark suite should also be assessed and agreed on by the whole community.

2.5. Previous interoperability evaluations

In the Semantic Web area, technology interoperability has been punctually evaluated. Some qualitative analyses have been performed in [OntoWeb, 2002] concerning ontology development tools, ontology merge and integration tools, ontology evaluation tools, ontology-based annotation tools, and ontology storage and querying tools; and in [Maynard *et al.*, 2007] concerning ontology-based annotation tools. These analyses provide information about the interoperability capabilities of the tools (such as the platforms where they run, the tools they interoperate with, or the data and ontology formats they manage), but they give no empirical studies to support their conclusions.

The only exceptions are the experiments carried out in the **Second International Workshop on Evaluation of Ontology-based Tools** (EON2003)¹. The central topic of this workshop was the evaluation of ontology development tools interoperability using an interchange language [Sure and Corcho, 2003].

¹<http://km.aifb.uni-karlsruhe.de/ws/eon2003/>

In this workshop, the participants were asked to model ontologies with their ontology development tools and to perform different tests for evaluating the import, export and interoperability of the tools.

The experiment had no restrictions on the interchange language, different languages (RDF(S), OWL, DAML, and UML) were used in different experiments, or on how to model the ontology to be interchanged, a natural language description of a domain was provided and each experimenter modelled the ontology in different ways.

The experiments performed in the EON2003 workshop are shown in table 2.8. In this table, we can observe the tools that are the origin and destination of the experiments, the interchange language used, and whether the experiment was circular or not, that is, if after exporting the ontology from the origin tool and importing it into the destination tool, the ontology is then exported again from the destination tool and imported into the origin tool. Further details about these experiments can be found in [Isaac *et al.*, 2003, Fillies, 2003, Corcho *et al.*, 2003b, Knublauch, 2003a, Calvo and Gennari, 2003]

Experiment	Origin	Destination	Interch. language	Circular
[Isaac <i>et al.</i> , 2003]	DOE	Protégé-2000	RDF(S)	X
	DOE	Protégé-2000	OWL	X
	DOE	OilEd	RDF(S)	X
	DOE	OilEd	OWL	X
	DOE	OntoEdit	RDF(S)	X
	DOE	OntoEdit	OWL	X
	DOE	WebODE	RDF(S)	X
	DOE	WebODE	OWL	X
[Fillies, 2003]	KAON	SemTalk	DAML	
	OilEd	SemTalk	DAML	
	OntoEdit	SemTalk	DAML	
	Protégé	SemTalk	RDF(S)	
	WebODE	SemTalk	RDF(S)	
[Corcho <i>et al.</i> , 2003]	WebODE	Protégé-2000	RDF(S)	X
[Knublauch, 2003a]	Protégé 2.0	Poseidon	UML	
	Protégé 2.0	OWL Validator	OWL	
[Calvo and Gennari, 2003]	Protégé 2.0	OilEd	RDF(S)	
	OilEd	Protégé 2.0	RDF(S)	X

Table 2.8: Experiments carried out at the EON2003 workshop.

Corcho's conclusions [2005] are extracted from the results of these experiments and from an analysis of the main features of RDF(S) and OWL. These conclusions are the following:

- RDF(S) and OWL allow representing the same knowledge in different ways, making knowledge exchange difficult. This is so because RDF(S)

and OWL ontologies can be serialized with different syntaxes (such as RDF/XML², Notation3³ or N-Triples⁴) and there are several ways to express knowledge in these syntaxes. Most of the existing tools can manage these syntaxes or use programming libraries to manage them. Nevertheless, some tools still use the serialized files directly, which can cause problems.

- The standard knowledge models of RDF(S), OWL Lite and OWL DL are not expressive enough to represent some of the knowledge that can be represented with traditional ontology languages and tools. Therefore, translations from more expressive knowledge models to these knowledge models usually involve knowledge losses.
- The translators to RDF(S) and OWL are usually written taking into account a specific language or tool. Two solutions have been adopted to avoid the loss of knowledge when translating from a more expressive model to a less expressive one:
 - To represent the knowledge that could be lost with annotation properties (such as *rdfs:comment*) using a specific structure for this information.
 - To extend the RDF(S) and OWL vocabularies with ad-hoc properties not defined in the specifications.
- Current translation systems in ontology development tools do still have many errors when exporting and/or importing RDF(S) and OWL.

2.6. Conclusions

When I was searching, unsuccessfully, for a software benchmarking methodology at the beginning of my thesis, I came across a collection of evaluation and improvement methodologies.

These methodologies, which belong to different areas (business management benchmarking, Software Measurement and Experimental Software Engineering), can be a starting point for a software benchmarking methodology because they share some common ways of solving their problems and possess some characteristics relevant to software benchmarking, such as considering evaluation as a continuous activity, having a company-wide perspective of software evaluation, and dealing with software experiments as a means for software evaluation.

However, this chapter does not present an exhaustive view of the different evaluation and improvement methodologies existing in these areas, since there are many of them. The methodologies here considered are those that are well-known and provide detailed descriptions of their processes, and have been taken as a starting point for the work performed.

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

³<http://www.w3.org/DesignIssues/Notation3>

⁴<http://www.w3.org/TR/rdf-testcases/#ntriples>

With regard to the problem of the interoperability of Semantic Web technologies, the EON2003 experiments were a first and valuable step toward evaluating this interoperability, as they highlighted the interoperability problems in the existing tools using the W3C recommended languages for ontology interchange.

Nevertheless, further evaluations of Semantic Web technology interoperability are required because

- Interoperability is a main problem for the Semantic Web that is still unsolved.
- The workshop experiments concerned only few tools and focused only on ontology development tools.
- Some experiments evaluated export functionalities, others, import functionalities, and only a few evaluated interoperability. Furthermore, interoperability from one tool to the same tool using an interchange language was not considered.
- No systematic evaluation was performed; each experiment used different evaluation procedures, interchange languages, and principles for modelling ontologies. Therefore, the results were not comparable and only specific comments and recommendations for each ontology development tool participating have been made.

Work objectives

3.1. Thesis goals and open research problems

The goal of this thesis is to advance the current state of the art in the Semantic Web area, first by providing **a methodology for benchmarking Semantic Web technologies**, and second by applying this methodology for **benchmarking the interoperability of Semantic Web technologies** using RDF(S) and OWL as interchange languages.

In order to achieve the first objective, the following (non exhaustive) list of open research problems must be solved:

- I. From a methodological perspective, there are at least three open problems:
 - The lack of a software benchmarking methodology. Although benchmarking methodologies abound in the business management community, there is no software benchmarking methodology, being it necessary to obtain the best practices used for developing Semantic Web technologies and to improve these technologies continuously.
 - The difficulty of using current evaluation and improvement methodologies with Semantic Web technologies. The benchmarking methodologies employed in the business management area, and the software evaluation and improvement methodologies employed in the Software Engineering area are general methodologies and they are not defined in detail. Therefore, it is difficult to use either of them in concrete cases within the Semantic Web area.
 - The absence of integrated methods and techniques supporting the complex task of benchmarking Semantic Web technologies. Several custom characterisations have been proposed under the perspective of the type of technology to be benchmarked (ontology development tools, ontology merge and alignment tools, ontology-based annotation tools, reasoners, etc.), but the approaches used to benchmark a type of technology are difficult to reuse and maintain since they are specific to that type of technology or even specific to a certain tool or set of tools.

- II. From a technological perspective, we lack tools that support benchmarking different types of Semantic Web technologies and that also support the different tasks that have to be performed in these benchmarking activities.

With regard to the second objective, benchmarking the interoperability of Semantic Web technologies, the following non exhaustive list of open research problems must be solved:

- I. The limits of the interoperability between current tools were unknown at the moment of starting this thesis.
- II. A method describing how to benchmark the interoperability of Semantic Web technologies using an interchange language has not yet been developed.
- III. No general benchmark suites can be reused for evaluating and benchmarking the interoperability of Semantic Web technologies using an interchange language.
- IV. Specific software support for evaluating and benchmarking the interoperability of Semantic Web technologies using an interchange language has not yet been developed.

3.2. Contributions to the state of the art

This thesis aims at giving solutions to the previous open research problems. Chapter 4 describes the solutions proposed for the first objective (the development of a benchmarking methodology for Semantic Web technologies) and chapters 5 and 6 present the solutions related to the second one (the benchmarking of the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages).

With regard to the first objective, the thesis presents new advances in the following aspects:

- C1. **A benchmarking methodology for Semantic Web technologies**, grounded on existing benchmarking methodologies and on practices in other areas, as general and open as possible so the methodology can cover the broad range of Semantic Web technologies. This methodology describes the benchmarking process with its sequential tasks, actors, inputs, and outputs. Such methodology has been validated by checking that it meets the necessary and sufficient conditions that every methodology should satisfy. Moreover, the methodology has been applied with successful results to different types of Semantic Web technologies in different scenarios and with different evaluation criteria.

The second objective of this thesis is to apply the previous benchmarking methodology with the purpose of benchmarking the interoperability of Semantic

Web technologies. The thesis here presented has contributed to the advance of the current state of the art by providing the following:

C2. A method for benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages. This method has been defined by instantiating the benchmarking methodology for Semantic Web technologies mentioned above and provides a framework for comparing the interoperability results of different types of Semantic Web tools. The method has been used in the two benchmarking activities performed in this thesis, namely, the RDF(S) and the OWL Interoperability Benchmarkings.

C3. The UPM Framework for Benchmarking Interoperability¹ (UPM-FBI) that includes all the resources needed for benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.

The UPM-FBI offers, as figure 3.1 shows, four benchmark suites that contain ontologies to be used in interoperability evaluations and two approaches for performing interoperability experiments (one manual and another automatic), each of them containing different tools that support the execution of the experiments and the analysis of the results.

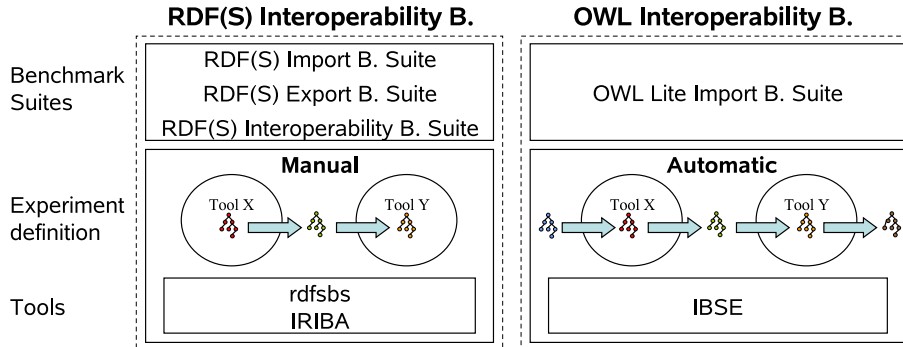


Figure 3.1: The UPM Framework for Benchmarking Interoperability.

The UPM-FBI provides the following mechanisms:

C3.1. Four benchmark suites for evaluating and benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages. These benchmark suites have been used as input for the experiments in the UPM-FBI. Three of them were defined to evaluate the interoperability using RDF(S) as the interchange language, namely, the *RDF(S) Import Benchmark*

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

Suite, the *RDF(S) Export Benchmark Suite*, and the *RDF(S) Interoperability Benchmark Suite*; and the fourth one was defined to evaluate the interoperability using OWL as the interchange language, the *OWL Lite Import Benchmark Suite*.

C3.2. A manual and an automatic approach for evaluating and benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.

In the manual approach, experiments are performed by accessing the tools manually, whereas in the automatic approach, the execution of the experiments and the analysis of the results are performed over the tools automatically.

C3.3. Software tools for evaluating and benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.

These software tools are needed to carry out the experiments in the UPM-FBI and have been developed having reusability in mind and, thus, they can be used in other evaluations. Two different tools support the manual approach, namely, the *rdfsbs* tool, which automates part of the experiments execution, and the *IRIBA*² web application, which provides an easy way of analysing the results³. One tool supports the automatic approach, the *IBSE*⁴ tool, which automates the experiments execution and the analysis of the results.

C4. A clear picture of the interoperability between different types of Semantic Web tools. Benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages has provided us with detailed information about the current interoperability of the tools participating in the benchmarking activities.

The **verification** of the goals stated in this thesis has taken place in the following environments:

- Most of the work performed in this thesis has been funded by the *Knowledge Web*⁵ European Network of Excellence (FP6-507482). Within Knowledge Web, benchmarking had a main role. The candidate was responsible for organizing and leading the benchmarking activities related to the interoperability of Semantic Web technologies, defining the experiments to perform, developing the benchmark suites and the software to use, and analysing the results of all the tools. Other participants executed the experiments over the tools participating in the benchmarking.

²<http://knowledgeweb.semanticweb.org/iriba/>

³This thesis does not deal with the IRIBA application as this is currently being developed by one undergraduate student.

⁴http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

⁵<http://knowledgeweb.semanticweb.org>

- The CICYT project *Infraestructura tecnológica de servicios semánticos para la web semántica*⁶ (TIN2004-02660) has funded the benchmarking of the performance, scalability and interoperability of the ontology development tool WebODE, with the aim of learning from its current performance and pushing the transfer of this tool to industry. This project has also funded the writing of this thesis.

3.3. Work assumptions, hypothesis and restrictions

This thesis is based on a set of **assumptions** that help explain the decisions taken for the development of the methodological and technological solutions and the relevance of the contributions presented. Such assumptions are listed below. As it can be observed, assumption A1 is related to the methodological objective of the thesis, whilst assumptions A2 and A3 deal with the interoperability benchmarking activities.

- A1. Semantic Web technologies (ontology development tools, ontology merge and alignment tools, ontology-based annotation tools, reasoners, etc.) can be empirically evaluated according to some quality criteria (interoperability, scalability, robustness, etc.) using different evaluation methods and tools.
- A2. Interoperability using an interchange language is the most common way used by Semantic Web tools to interchange ontologies.
- A3. RDF(S) and OWL are the languages commonly used by Semantic Web tools to interchange ontologies.

Once the assumptions have been identified and presented, the set of **hypotheses** of this thesis are described. Hypotheses H1 to H3 are related to the methodological objective of the thesis, whilst hypotheses H4 to H8 deal with the interoperability benchmarking activities.

- H1. Evaluation and benchmarking methodologies in other areas are general enough to be taken as a starting point to develop a benchmarking methodology for Semantic Web technologies.
- H2. The processes for benchmarking different Semantic Web technologies have common tasks for all the types of technologies.
- H3. The benchmarking methodology for Semantic Web technologies proposed in this thesis can be used to evaluate and improve different types of Semantic Web technologies.

⁶<http://droz.dia.fi.upm.es/servicios/>

- H4. The interoperability of the different types of Semantic Web technologies using an interchange language can be evaluated using a common method.
- H5. Interoperability using an interchange language highly depends on the knowledge models of the tools. Interoperability is better when such knowledge model is similar to that of the interchange language.
- H6. The interoperability of the tools depends not only on the defects that the tools may have but also on development decisions taken by their developers.
- H7. The interoperability of the tools improves after benchmarking them.
- H8. The cost of performing interoperability experiments and the quality of their results depends on the method followed to perform the experiments and to analyse their results.

Finally, the following set of **restrictions** defines the limits of the contributions of this thesis and allows determining future research objectives. These restrictions delimit the research problem and allow the incremental improvement of research. Restrictions R1 to R3 are related to the methodological objective of the thesis, whilst restrictions R4 to R9 deal with the interoperability benchmarking activities.

- R1. The benchmarking methodology for Semantic Web technologies proposed in this thesis only deals with Semantic Web technology evaluation, not with ontology evaluation.
- R2. The benchmarking methodology for Semantic Web technologies proposed in this thesis only deals with such technologies, it does not contemplate other types of technologies.
- R3. The benchmarking methodology for Semantic Web technologies proposed in this thesis only deals with software products, not with software processes or services.
- R4. This thesis does not provide technological support to the general software benchmarking process; the software provided just solves the specific scenario of the benchmarking of the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages.
- R5. Of the different evaluation criteria that can be used to evaluate and benchmark Semantic Web technologies, e.g., efficiency, scalability, interoperability, robustness, etc., this thesis only contemplates interoperability using RDF(S) and OWL as interchange languages.
- R6. This thesis only contemplates interoperability at the information level; interoperability at the system level has not been taken into account.

- R7. This thesis only tackles the problem of interoperability between Semantic Web technologies using RDF(S) and OWL as interchange languages. Therefore, methods, tools or benchmark suites for other approaches to the interoperability problem are not provided.
- R8. The interoperability evaluation performed in the benchmarking activities is not exhaustive; interoperability is measured using only ontologies with simple combinations of components. Furthermore, the full knowledge models of the tools or of the interchange languages have been disregarded when defining the benchmark suites and only a subset of them has been taken into account.
- R9. The interoperability evaluation performed in the benchmarking activities only contemplates ontology interchanges from one origin tool to a destination one. Cyclic interoperability experiments from one origin tool to a destination one and then back to the origin tool, or chained interoperability experiments from one origin tool to an intermediate one and then to a third one, have been disregarded.

Benchmarking methodology for Semantic Web technologies

This chapter presents a methodology for benchmarking Semantic Web technologies. First, section 4.1 presents the design principles taken into account when defining the methodology and section 4.2 describes the process followed to define it. Then, section 4.3 details the methodology by describing its actors, process and tasks. And, finally, section 4.4 shows how two benchmarking activities (the RDF(S) Interoperability Benchmarking and the OWL Interoperability Benchmarking) were organized and carried out following the proposed methodology.

In this thesis the idea of benchmarking has been adopted from the business management community [Camp, 1989, Spendolini, 1992], and software benchmarking is defined as a collaborative and continuous process for improving software products, services and processes by systematically evaluating and comparing them to those considered the best.

From this definition derives the **purpose** of the benchmarking methodology for Semantic Web technologies, which is to provide a comprehensive process when benchmarking Semantic Web software, identifying the sequence of tasks that compose the process, its inputs and outputs and the actors that participate in these tasks.

The **use scenarios** of this methodology always involve a number of existing types of Semantic Web technologies (those presented in chapter 1, which are related to ontology development, ontology management, instance generation, semantic information storage, querying and reasoning, semantic information access, programming and development, and application integration). These technologies can be either in development or in production stages, being the main scenarios in which the methodology can be presented the following:

- *Improvement of a group of Semantic Web tools.* One community (research, industrial, user, or a mix of the previous) is interested in evaluating and

improving a group of tools by applying the best practices learnt from the others. The tools can be of the same or of different type.

- *Evaluation of a group of Semantic Web tools.* One community (research, industrial, user, or a mix of the previous) is interested in obtaining a detailed comparison of a group of tools and of the practices carried out when developing them. These tools can be of the same or of different type.
- *Improvement of a Semantic Web tool.* In this scenario, a software developer organization (research or industrial) is interested in continuously improving its own tool.
- *Assessment of a benchmarking process.* Benchmarking is carried out over Semantic Web tools and the benchmarking partners are interested in assessing, at a specific point in time, which is the actual progress of benchmarking.

4.1. Design principles

The design principles applied when developing the benchmarking methodology were extracted from the conditions that every methodology should satisfy [Paradela, 2001]. These conditions can be classified into necessary conditions (also called formal conditions), which are independent of the domain where the methodology is applied and are common to any methodology, and into sufficient conditions (also called material conditions), which are specific to each domain where the methodology is used, in our case, to the domain of benchmarking Semantic Web technologies.

The **necessary conditions** of any methodology are those defined by Paradela [2001], and they are the following:

- *Completeness.* The methodology must consider all the cases presented, regardless of who presents the case, and of their degree of difficulty, typology, etc.
- *Efficiency.* The methodology must be an efficient procedure, in the sense that it must be independent of the person who uses it.
- *Effectiveness.* The methodology must be effective and able to solve adequately all the cases that, having a solution, are presented.
- *Consistency.* The methodology must produce the same results for the same problem, independently of who carries it out.
- *Responsiveness.* The methodology must consider the “what”, “who”, “why”, “when”, “where”, and “how” of every task.
- *Advertency.* The methodology must consider the general methodological rules of Descartes’ Cartesian Method: *evidence* (never to accept anything

for true which I do not clearly know to be such), *analysis* (to divide each of the difficulties under examination into as many parts as possible, and as might be necessary for its adequate solution), *synthesis* (to conduct thoughts by commencing with objects the simplest and easiest to know and ascending little by little to the knowledge of the more complex), and *test* (to make enumerations so complete, and reviews so general, that I might be assured that nothing was omitted).

- *Finiteness*. Both the number of the elements composing the methodology and the number of tasks must be finite and they should consume a short period of time.
- *Discernment*. The methodology must be composed of a small number of structural, functional and representational components.
- *Environment*. The methodology must be classified into one of two main groups: scientific or technological.
- *Transparency*. The methodology must be like a white box, allowing us to know in every moment which is the task being processed, what is being performed, who is performing it, etc.

The **sufficient conditions** of the benchmarking methodology are those specific to the domain of benchmarking Semantic Web technologies, and they are the following:

- *Benchmarking-oriented*. The methodology must be focused on producing a continuous improvement of the tools and on obtaining the best practices used when developing these tools. However, it does not have to provide thorough details of other tasks related to Experimental Software Engineering, Software Configuration Management, organization management, or staff training.
- *Grounded on existing practices*. The methodology must be grounded on existing methodologies and practices in the areas of benchmarking, Experimental Software Engineering and Software Measurement so as to facilitate its understanding and use in different organizations.
- *Collaborativeness*. The methodology must permit carrying out benchmarking in a collaborative way and should be flexible enough so that any organization interested in participating in the benchmarking and in producing consensual outcomes could do it. This collaboration will lend credibility to the benchmarking results.
- *Openess*. The methodology must not limit the software entities to be considered in benchmarking (products, services or processes), nor the phase of the software life cycle when benchmarking is performed, nor the people responsible for carrying out benchmarking. This openness will permit applying the methodology in a broad range of scenarios.

- *Usability.* The methodology must be easy to understand and learn, and the effort needed to use the methodology must be minimal independently of the complexity of the benchmarking.

4.2. Research methodology

This section describes the process followed to develop the benchmarking methodology for Semantic Web technologies.

For describing the methodology, the definitions of phase, process and task set out by the IEEE [IEEE, 2000] have been strictly followed:

- *Phase.* A distinct part of a process in which related operations are performed.
- *Process.* A sequence of tasks, actions, or activities, including the transition criteria for progressing from one to the next, that bring about a result.
- *Task.* The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members.

The steps followed during the development of the benchmarking methodology, as figure 4.1 shows, were

1. To select relevant processes from other research areas. In this case, from the benchmarking, Experimental Software Engineering and Software Measurement areas.
2. To identify the main tasks of the processes selected by choosing the tasks that are considered in most of the processes of each area.
3. To adapt and complete these tasks in order to cover the requirements for successful benchmarking.
4. To analyse task dependencies in order to define task order.

Details of the work performed in each of these steps can be seen in the next sections.

4.2.1. Selection of relevant processes

Relevant processes from the areas of benchmarking, Experimental Software Engineering and Software Measurement were selected with the goal of obtaining a clear description of the common tasks that are performed in these areas' processes. Some other processes can be found in the literature, being their main tasks the same or similar to those already identified.

The processes selected, described in chapter 2 and classified into areas, are the following:

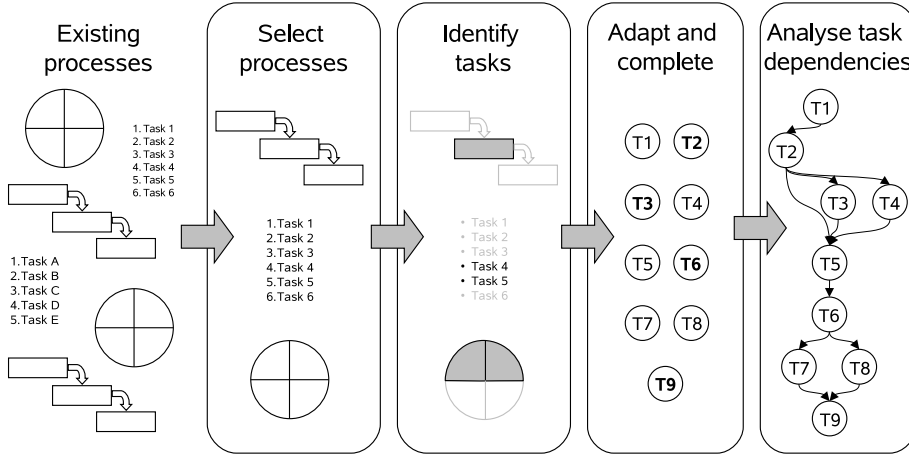


Figure 4.1: Steps followed during the development of the methodology.

■ Benchmarking area

- Camp's [1989] benchmarking process.
- Andersen and Pettersen's [1996] benchmarking wheel.
- The APQC benchmarking methodology [Gee *et al.*, 2001].

■ Experimental Software Engineering area

- Basili et al.'s [1986] experimentation process.
- Pfleeger's [1995] experimentation steps.
- Wohlin et al.'s [2000] experimentation process.
- Kitchenham et al.'s [2002] experimentation guidelines.

■ Software Measurement area

- Grady and Caswell's [1987] software metrics process.
- Goodman's [1993] software metrics process.
- McAndrews's [1993] software measurement process.

4.2.2. Identification of the main tasks

Chapter 2 described in detail the tasks that compose each of the previous processes. From these processes, some common tasks that are considered in most of the processes of each area were selected, as can be seen in table 2.3 (benchmarking), table 2.4 (Software Measurement), and table 2.7 (Experimental Software Engineering).

Table 4.1 shows the selected common tasks classified by area, aligning similar tasks in different areas.

As can be seen, some of the tasks are similar in the three areas, but as each area addresses different situations some differences always appear. For example, experimentation processes focus more on the experimentation-related tasks, whereas the search for partners only occurs in benchmarking, and improvement is only considered in benchmarking and software metrics.

The tasks that are common to these three areas were selected and those that are specific to benchmarking were incorporated. Therefore, the initial list of tasks includes the following:

- To define the subject.
- To define the goals.
- To form the team.
- To identify partners.
- To define metrics.
- To define data collection methods.
- To execute the experiment.
- To analyse data.
- To interpret results.
- To improve.
- To document results.
- To disseminate results.

4.2.3. Task adaption and completion

As the above selected tasks did not cover all the requirements for successful benchmarking that could guarantee a continuous improvement in Semantic Web technologies, new tasks were included in the benchmarking process that completed the coverage of these requirements, thus ensuring improvement, continuity, planning, and consensus. These tasks are

- The involvement of the organization management is needed in any activity that implies organizational changes or resource expenditure. Therefore, a task related to this involvement was added, which ensured improvement in the software.
- Benchmarking is a continuous task. Therefore, two tasks were included, the first related to the monitorization of the improved software and the second, to the recalibration of the same benchmarking process for future benchmarking iterations.

Benchmarking	Software Measurement	Experimental Software Engineering
Identify the process to be benchmarked	Identify the needs for measurement	Define the subject
	Define organization objectives	Define the purpose
		Define the perspective
		Define the context
Form and train the benchmarking team	Assign responsibilities	
Analyse and document the current process		
Define criteria for benchmarking partners		
Identify potential partners	Research external products	
Contact the potential partners and enlist them		
		Define hypothesis
		Design the experiment
		Define the criteria
		Define the metrics
Define method and tools for data collection	Define metrics to collect	Define the method for obtaining the measures
	Define data collection methods	
	Define storage/analysis/feedback methods	
	Establish training mechanisms	
	Build the metrics database	
Collect data	Collect data	Prepare the subject for the experiment
Determine the current performance gaps	Analyse data	Execute the experiment
Identify the causes for the gaps		Validate the data obtained
Communicate findings and gain acceptance		Analyse the data collected
Formulate strategy to close the gaps		Interpret results
Develop the implementation plan		
Implement the plan	Change	
Monitor and report progress		
Document the study	Disseminate results	Document the experiment
		Present results

Table 4.1: Common tasks identified in the relevant processes.

- As the methodology is intended to be used by several groups of people, several tasks were added and dedicated to the planning of the benchmarking in different times: when organizing it, when carrying out the experiment, and when improving the software in each organization.
- Benchmarking requires consensus and communication between different groups of people, both in the same and in different organizations. Therefore, one of the tasks added was devoted to the writing of the benchmarking proposal, which will be used as a reference along the process.

Once the list of required tasks was selected, these tasks were completed by identifying their inputs, outputs and participants. Specific techniques for carrying out each task were not regarded because one of the requirements was to obtain a general methodology. Each benchmarking requires to carry out different tasks, and many works in literature deal with the different techniques used in the tasks, such as team and process management or software evaluation.

4.2.4. Analysis of task dependencies

The last step, once the tasks and their inputs, outputs and participants were identified was to define the order in which these tasks are to be performed. To arrange the order of the tasks, a task A was considered to be previous to a task B if the output of the task A is needed as an input in task B.

For example, the *Experiment execution* task requires as input the definition of the experiment. Therefore, since the definition of the experiment is an output of the *Experiment definition* task, this task must be performed before the *Experiment execution* task.

An ordered benchmarking process with parallel and sequential tasks was obtained. To simplify this process, some parallel tasks were merged into one (e.g., the *Define subject* and *Define metrics* tasks, and the *Analyse data* and *Interpret results* tasks) because they were performed by the same actors and the tasks and their outputs were highly coupled.

4.3. Benchmarking methodology

4.3.1. Benchmarking actors

The tasks of the benchmarking process are carried out by different actors according to the kind of roles that must be performed in each task. This section presents the different kind of actors involved in the benchmarking process.

- *Benchmarking initiator.* The benchmarking initiator is the member (or members) of an organization who performs the first tasks of the benchmarking process. His work consists in preparing a proposal for carrying out the benchmarking and obtaining the approval from the organization management to perform it.

- *Organization management.* The organization management plays a key role in the benchmarking process, as it must approve benchmarking and support the changes that result from it. It must also assign resources to the benchmarking and integrate the benchmarking planning into the organization planning.
- *Benchmarking team.* Once the organization management approves the benchmarking proposal, it forms the benchmarking team, which consists of the members of the organization responsible for performing most of the remaining benchmarking tasks.
- *Benchmarking partners.* The benchmarking partners are the organizations that participate in the benchmarking. All the partners must agree on the steps to follow during the benchmarking and their needs must be taken into account.
- *Tool developers.* The developers of the tool considered for the benchmarking are those that will implement the necessary changes in the tool in order to improve it, taking into account the benchmarking recommendations. Some of the developers may also belong to the benchmarking team and, in this case, care must be taken to minimise bias.

4.3.2. Benchmarking process

The benchmarking process defined in this methodology is a continuous process that should be performed indefinitely in order to obtain a continuous improvement in the tools participating in the benchmarking.

Figure 4.2 shows the main phases of the benchmarking process. This process is composed of a benchmarking iteration that is repeated forever.

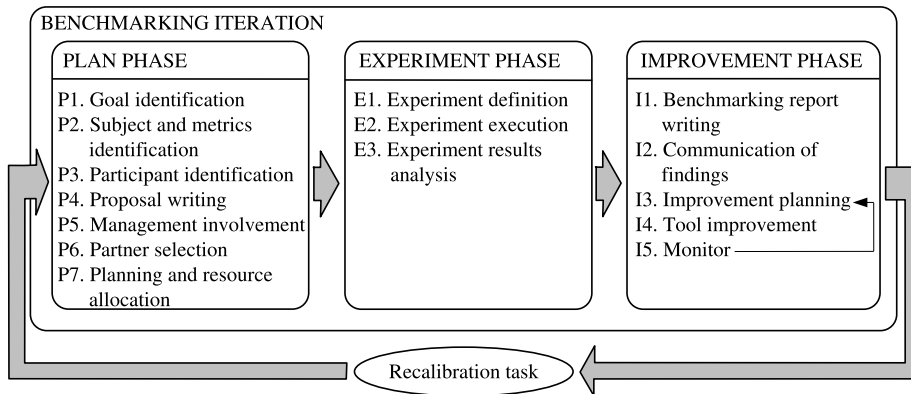


Figure 4.2: The benchmarking process.

Each benchmarking iteration is composed of three phases (*Plan*, *Experiment*, and *Improvement*) and ends with a *Recalibration* task. The main goals of these phases are the following:

- *Plan phase*. It comprises the set of tasks that must be performed in order to prepare the proposal for benchmarking, to find other organizations that want to participate in the benchmarking, and to plan the benchmarking.
- *Experiment phase*. It comprises the set of tasks in which the experiment over the different tools considered is performed.
- *Improvement phase*. It comprises the set of tasks where the results of the benchmarking process are produced and communicated to the benchmarking partners, and where the improvement of the different tools is performed in several improvement cycles.

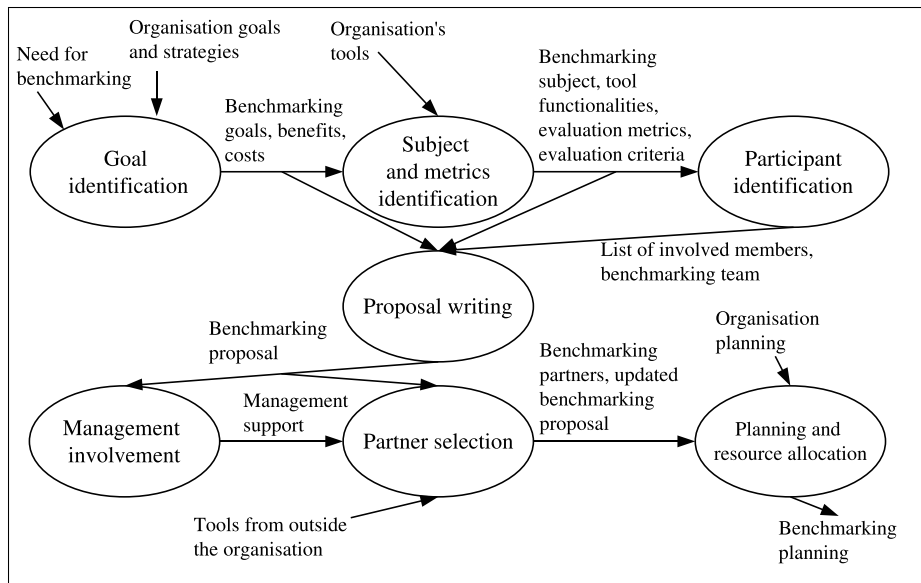
These three phases of the benchmarking process are described in the following sections, where a definition of the tasks that constitute them, the actors that perform these tasks, its inputs, and its outputs are provided.

While the three phases mentioned above are devoted to tool improvement, the goal of the *Recalibration* task is to enhance the benchmarking process itself. This task is described at the end of the chapter.

4.3.3. Plan phase

The *Plan* phase comprises the set of tasks that must be performed to prepare the proposal for benchmarking, find other organizations willing to participate in the benchmarking, and plan the benchmarking. These tasks and its interdependencies, shown in figure 4.3, are the following:

- P1. Goal identification.
- P2. Subject and metrics identification.
- P3. Participant identification.
- P4. Proposal writing.
- P5. Management involvement.
- P6. Partner selection.
- P7. Planning and resource allocation.

Figure 4.3: *Plan* phase of the benchmarking process.

P1. Goal identification

Actors	
Benchmarking initiator	
Inputs	Outputs
Need for benchmarking	Benchmarking goals
Organization goals and strategies	Benchmarking benefits
	Benchmarking costs

The benchmarking process always starts in an organization in which one of its members is aware of the need for benchmarking. This need varies across organizations and is highly related to the desired goals of the benchmarking process.

When benchmarking is carried out in an organization, the main goals are the following [Sole and Bist, 1995, Wireman, 2003, Kraft, 1997]:

- To assess the performance of the organization's products and processes over the time.
- To improve the quality of the organization's products and processes.
- To compare the organization's products and processes with those of the best organizations and to close the performance gap.
- To obtain a deep understanding of the practices that create superior products and processes.

- To establish standards set by the industry or the best organizations, or to create standards after analysing the best organizations.
- To increase the customer's satisfaction with the organization's products.

Those members of the organization who are aware of the need for benchmarking take the role of benchmarking initiators. The benchmarking initiator will be the one who carries out the first tasks of the process.

During this task, the benchmarking initiator must identify the benchmarking goals, which are usually in concordance with the organization goals and strategies. He must also identify the benefits that the benchmarking process will bring to the organization, and the costs of performing benchmarking.

P2. Subject and metrics identification

Actors	
Benchmarking initiator	
Inputs	Outputs
Benchmarking goals	Benchmarking subject
Benchmarking benefits	Tool's relevant functionalities
Benchmarking costs	Evaluation metrics
Organization's tools	Evaluation criteria

The goal of this task is to identify which of the tools developed in the organization will be benchmarked. This includes identifying the tool functionalities relevant for the study and the evaluation metrics and criteria that will be adopted to assess these functionalities. The benchmarking subject, functionalities, metrics, and criteria should be those whose improvement would significantly benefit the organization.

On the other hand, the benchmarking initiator must perform an analysis of the tools developed in the organization in order to understand them. This analysis must be documented by providing a description of the tools and their weaknesses and functionalities that need improvement.

Then, the benchmarking initiator must select which of these tools will be the benchmarking subject and which functionalities and criteria will be considered according to

- The analysis of the tools developed in the organization.
- The benchmarking goals, benefits, and costs identified in the previous task.
- Other factors viewed as critical in the organization, such as quality requirements, end user needs, etc.

P3. Participant identification

Actors	
Benchmarking initiator	
Inputs	Outputs
Benchmarking subject Tool's relevant functionalities Evaluation metrics Evaluation criteria	List of involved members Benchmarking team

Additionally, the benchmarking initiator must identify and contact the members of the organization that are involved with the tool and functionalities selected. This group of people can be made up of managers, developers, ontology engineers, end users, etc. Other relevant participants from outside the organization, such as customers or consultants, can also form part of this group.

The benchmarking initiator must also select the members of the benchmarking team that will be in charge of performing most of the remaining benchmarking tasks. Quite frequently, the benchmarking initiator is a member of the benchmarking team.

The benchmarking team must be composed of the organization members whose work and interest are related to the type of tools that will be benchmarked. These members should have a thorough understanding of these tools and experience of working with them.

The benchmarking team should be small, and its members must be aware that benchmarking is time consuming, so they must dedicate much of their time to it.

Finally, the benchmarking team must have their responsibilities clearly defined and should also be trained in the different tasks to be performed in the remaining of the benchmarking process.

P4. Proposal writing

Actors	
Benchmarking initiator Benchmarking team	
Inputs	Outputs
Benchmarking goals Benchmarking benefits Benchmarking costs Benchmarking subject Tool's relevant functionalities Evaluation metrics Evaluation criteria List of involved members Benchmarking team	Benchmarking proposal

In this task, the benchmarking team (and the benchmarking initiator if he does not belong to the team) must write a document with the benchmarking proposal. The proposal will be used as a reference along the benchmarking process and should include all the relevant information.

It must also include all the information identified in the previous benchmarking tasks with an approximate description of the benchmarking process, and a very detailed description of the benchmarking costs with an estimation of the resources needed in the benchmarking process: people, equipment, travel, etc.

Additionally, the benchmarking team must take into consideration when writing the benchmarking proposal that such proposal has different intended readers: organization management, organization developers, members of partner organizations, and the benchmarking team themselves. Therefore, the proposal must be clear and understandable to all readers. It must include

- The description of the benchmarking process.
- The benchmarking goals.
- The benchmarking benefits.
- The benchmarking costs.
- The benchmarking subject.
- The tool's relevant functionalities.
- The evaluation metrics and criteria for these functionalities.
- The list of members involved.
- The members of the benchmarking team.
- The resources needed in the benchmarking process.

P5. Management involvement

Actors	
Benchmarking initiator Organization management	
Inputs	Outputs
Benchmarking proposal	Management support

In this task, the benchmarking initiator must present the benchmarking proposal to the organization management. He must inform the organization management of the benchmarking process, its goals, its benefits, and its costs. He must also inform of which components of the organization are, and will be, involved in the process, i.e., which tool will be benchmarked, who are the members involved in the benchmarking process, who will be part of the benchmarking team, etc.

This task is of great importance because it requires that the management gives its approval to continue with the benchmarking process. On the other hand, the organization management must commit enough resources for carrying out the benchmarking and support the changes that, in a near future, will be implemented, either in the tool or in the organization processes that affect the tool.

P6. Partner selection

Actors	
Benchmarking team Benchmarking partners	
Inputs	Outputs
Benchmarking proposal Management support Tools developed outside the organization	Benchmarking partners Updated benchmarking proposal

Once benchmarking has the organization management support, the benchmarking team must select the different tools to be dealt with in the benchmarking activity.

To do this, the benchmarking team must investigate and identify the tools that are comparable to the tool selected as the benchmarking subject. The team must also collect and analyse information of these tools and of the organizations that develop them.

Then, the team must establish the criteria that identify which of the tools are suitable for benchmarking. According to these criteria, the team must select from the comparable tools those that will be analysed in the benchmarking study.

These criteria are varied and they can be, for example, the relevance of a tool in a community or in the industry, how this tool utilizes the latest technological tendencies, how many people use such a tool, the public availability of the tool, etc. Furthermore, in order to obtain better results with benchmarking, the chosen tools should be those considered the best.

When the benchmarking team has selected the tools, they must contact any member of the organizations that develop each of these tools to learn whether these organizations are interested in benchmarking.

If an organization that develops a tool is not interested in benchmarking, the team can make contact with others to see if they want to participate by assessing that tool, even if they are not the developers. For instance, tool users could participate and align the benchmarking goals and scope with their own interests.

The organizations willing to participate become benchmarking partners. Then, they will also have to form a benchmarking team and take the benchmarking proposal to their own organization management for approval.

During the course of this task, the proposal will be modified to include the partners' opinions and needs. The modifications introduced will result in an updated proposal, which should be used in the rest of the benchmarking. If the modifications are significant, the proposal will be presented again to each partner organization management for approval.

P7. Planning and resource allocation

Actors	
Benchmarking teams Management of the organizations	
Inputs	Outputs
Benchmarking partners Updated benchmarking proposal Organization planning	Benchmarking planning

In this task, the benchmarking teams and the managers from each partner organization must do some detailed planning for the rest of the benchmarking process and reach a consensus on it. This planning must then be considered and integrated into each organization's planning.

The benchmarking planning must take into account the considerable effort that will be devoted and the organization resources to be allocated, such as people, computers, travel, etc.

4.3.4. Experiment phase

The *Experiment* phase comprises the set of tasks in which the experiments with the different tools considered in the benchmarking is performed. These tasks and their interdependencies, shown in figure 4.4, are the following:

- E1. Experiment definition.
- E2. Experiment execution.
- E3. Experiment results analysis.

E1. Experiment definition

Actors	
Benchmarking teams	
Inputs	Outputs
Updated benchmarking proposal Benchmarking planning	Experiment definition Experimentation planning

In this task, the benchmarking teams of each partner organization must define the experiment that will be performed in each of the tools involved in the benchmarking.

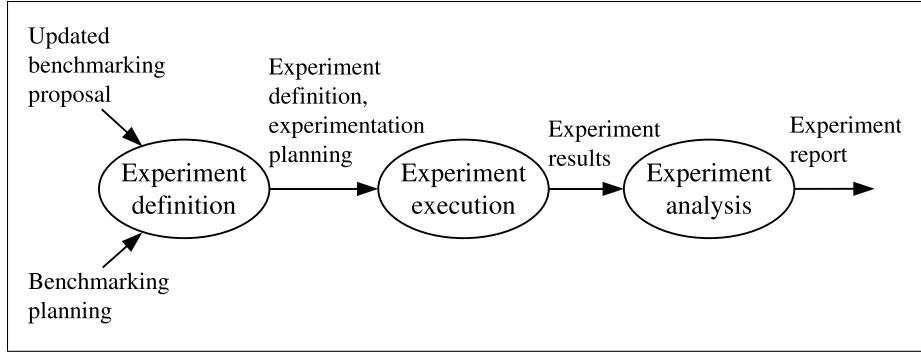


Figure 4.4: *Experiment* phase of the benchmarking process.

The experiment, which can change between different benchmarking iterations, must be defined according to the intended benchmarking goals and must evaluate the selected functionalities of the tools according to their corresponding criteria, as stated in the benchmarking proposal.

The experiment must provide objective and reliable data on the tools, not just on its performance but also on the reasons of its performance, and must be defined taking into account its future reuse.

The benchmarking teams, on the other hand, must devise the planning to be followed during experimentation. Such planning should be decided according to the benchmarking planning devised in the previous task.

The experiment definition and its planning must be communicated to all the partners, who must agree on it.

E2. Experiment execution

Actors	
Benchmarking teams	
Inputs	Outputs
Experiment definition Experimentation planning	Experiment results

According to the experimentation planning devised in the previous task, the benchmarking teams must perform the experiments on their tools.

Then, the data obtained from all the experiments must be compiled, documented, and expressed in a common format to facilitate its future analysis.

E3. Experiment results analysis

Actors	
Benchmarking teams	
Inputs	Outputs
Experiment results	Experiment report

In this task, the benchmarking teams must analyse the results of the experiments. This analysis involves comparing the results of the experiments performed on each tool and the practices that led to these results.

During the analysis, the teams must detect and document any significant differences observed in the results of the tools, identifying the practices leading to these different results. The teams must also attempt to identify if, between the practices found, some of them can be considered best practices.

When the analysis of the experiment results ends, the teams must write a report with all the findings, namely, results of the experiment, differences in the results, practices and best practices found, etc.

4.3.5. Improvement phase

The *Improvement* phase is composed of the set of tasks where the results of the benchmarking process are produced and communicated to the benchmarking partners and the improvement of the different tools is performed in several improvement cycles. In these improvement cycles, improvement is planned, carried out and monitored. These tasks and their interdependencies, shown in figure 4.5, are the following:

- I1. Benchmarking report writing.
- I2. Communication of findings.
- I3. Improvement planning.
- I4. Tool improvement.
- I5. Monitor.

I1. Benchmarking report writing

Actors	
Benchmarking teams	
Inputs	Outputs
Updated benchmarking proposal Experiment report	Benchmarking report

In this task, the benchmarking teams must write the report of the benchmarking. Whereas the experiment report provides technical details and results

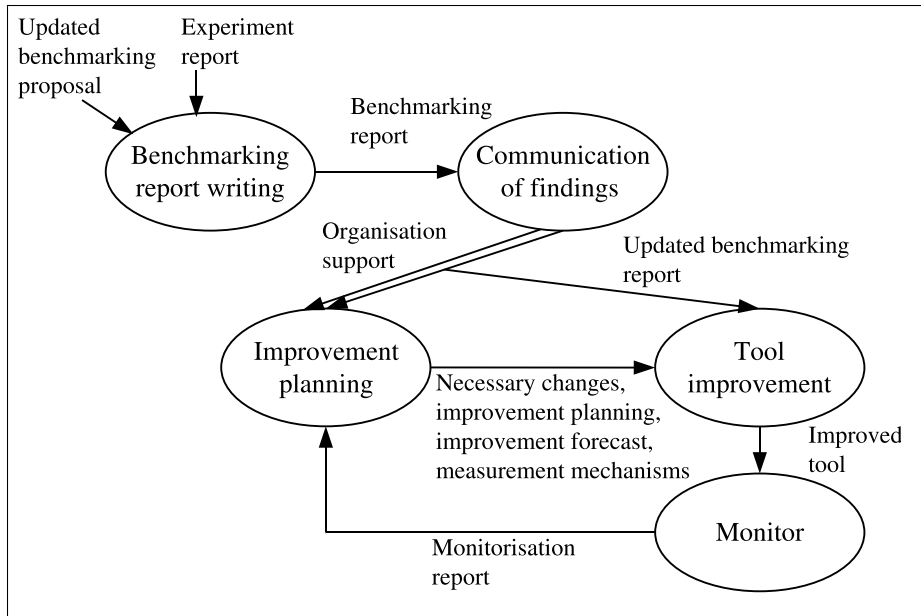


Figure 4.5: *Improvement* phase of the benchmarking process.

on the experiment, the benchmarking report is intended to provide an understandable summary of the process carried out. Thus, the benchmarking report should be written having in mind different audiences: managers, benchmarking teams, developers, etc.

The report should include an explanation of the benchmarking process followed and all the relevant information from the updated version of the benchmarking proposal; it should also include the results and conclusions of the experiments present in the experiment report, highlighting the best practices found during the experimentation and including any best practices found in the community.

The report must also contain the recommendations of the benchmarking team for improving the tools according to the experiment results, the practices found, and the community best practices.

The goal of the benchmarking report is not to provide a ranking of the tools, but to provide the practices and the best practices found in the benchmarking and to give improvement recommendations.

I2. Communication of findings

Actors	
Benchmarking teams	
Inputs	Outputs
Benchmarking report	Updated benchmarking report Organization support

In this task, the benchmarking teams must communicate the results of the benchmarking study to their organizations, and particularly to all the members involved and identified when planning the benchmarking.

This communication should be in the form of meetings, which should be held for one or more partner organizations. The goals of any benchmarking team in these meetings are twofold:

- To obtain feedback from the members concerned of the benchmarking process, results, and improvement recommendations.
- To obtain support and commitment from the organization members in order to implement the improvement recommendations in the tool.

Any feedback received during the communications of these findings must be collected, documented, and analysed. Finally, this analysis may result in having to review the work done and to update the benchmarking report.

I3. Improvement planning

Actors	
Benchmarking teams Management of the organizations Tool developers	
Inputs	Outputs
Updated benchmarking report Monitoring report Organization support	Necessary changes Improvement planning Improvement forecast Measurement mechanisms

The last three tasks of the *Improvement* phase (*Improvement planning*, *Tool improvement*, and *Monitor*) form a cycle that must be performed by each organization separately. It must be added that it is from these tasks where each organization benefits from the results obtained in benchmarking.

In this task, the benchmarking team, the managers and the tool developers of each partner organization must identify, from the benchmarking report and the monitoring reports, which are the changes required to obtain an improved tool. They must also foresee which improvement will be achieved after performing these changes.

Both the organization management and the benchmarking team must provide the organization with mechanisms that ensure the accomplishment of the improvements.

Additionally, the benchmarking team should provide tool developers with mechanisms for measuring tool functionalities. These mechanisms can be obtained or adapted from the tools and input data used in the *Experiment* phase.

Then, they must decide the planning for improving the benchmarked tool and reach a consensus. This planning should be reckoned with and integrated into each organization planning.

The improvement planning should contemplate the time and the organization resources (people, computers, travel, etc.) devoted to improvement activities.

I4. Tool improvement

Actors	
Tool developers	
Inputs	Outputs
Updated benchmarking report Necessary changes Improvement planning Improvement forecast Measurement mechanisms	Improved tool

In this task, the developers of each benchmarked tool should implement the necessary changes in order to achieve the desired results.

But before implementing any changes, the tool developers must assess the actual state of the tool using the measurement mechanisms provided by the benchmarking team in the *Improvement planning* task. Then, after implementing the necessary changes, the developers must measure the tool again and compare the results with both the results obtained before implementing the changes and the improvement foreseen.

I5. Monitor

Actors	
Benchmarking team Tool developers	
Inputs	Outputs
Improved tool	Monitoring report

In each organization, the benchmarking team must provide tool developers with means for monitoring the organization's tool, such as monitoring tools, benchmark suites, etc.

Tool developers must periodically monitor the tool and write a report with the results of this monitoring.

These monitoring results may show the need for new improvements in the tool and the beginning of a new improvement cycle, which means having to perform again the two previously mentioned tasks: *Improvement Planning* and *Tool improvement*.

4.3.6. Recalibration task

Actors	
Benchmarking team	
Inputs	Outputs
Benchmarking process	Improved benchmarking process
Lessons learnt	

The recalibration task is performed at the end of each benchmarking iteration. In this task, the benchmarking team must recalibrate the benchmarking process using the lessons learnt while performing such process. Thus, the organization improves not just the tools, but also the benchmarking process. This recalibration is needed because both the tools and the organizations evolve over the time.

4.4. Organizing the benchmarking activities

The RDF(S) and the OWL Interoperability Benchmarkings, presented in chapter 3, were organized and carried out following the benchmarking methodology for Semantic Web technologies described above, which provides the general guidelines that have to be adapted to each case.

This section includes the instantiation of this methodology in both benchmarking activities from the beginning of the benchmarking to the *Experiment definition* task of the *Experiment* phase. The content of the rest of the section is valid for the two benchmarking activities since these two activities have similar goals and scope and the tasks followed in the *Plan* phase are also similar. When needed, the results of the tasks in each of the benchmarking activities are clearly differentiated.

The following chapters present the instantiation of the methodology in the other tasks of the RDF(S) Interoperability Benchmarking (chapter 5) and of the OWL Interoperability Benchmarking (chapter 6). The chapters contain a complete definition of the experiments performed (including the benchmark suites and the evaluation software used), information on how the experiments were executed, and a detailed analysis of the results.

4.4.1. Plan phase

The author took the role of the benchmarking initiator and was in charge of organizing and defining the benchmarking, carrying out the first tasks of its process.

Goals identification

According to the benchmarking methodology for Semantic Web technologies, the first task to perform is to identify the benchmarking goals, benefits and costs.

Our goal was **to evaluate and improve the interoperability of Semantic Web technologies** using, in one case, RDF(S) and, in the other case, OWL as the interchange language.

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

In the RDF(S) Interoperability Benchmarking, the scope of the benchmarking was limited to one type of technology, namely, ontology development tools. However, though the scope was limited, the benchmarking was intended to be general enough to allow other types of tools to participate. In the OWL Interoperability Benchmarking, on the other hand, the scope was broadened and considered any type of Semantic Web technology.

The **benefits** pursued through our goal, which will be commented below, are related to the expected outcomes of the benchmarking and involve different communities dealing with Semantic Web tools, namely, the research community, the industrial community, and tool developers. Such benefits are

- To create consensual processes and mechanisms for evaluating the interoperability of these tools.
- To produce user and developer recommendations on the interoperability of these tools.
- To acquire a deep understanding of the practices used to develop these tools and of how the practices used affect their interoperability.
- To extract from these practices those that can be considered best practices when developing the tools.

Most of the **costs** of the benchmarking go to the human resources needed to organize the benchmarking, to define the experimentation process, to perform the experiments on the tools, and to analyse the results. Other minor expenditure goes to travelling and computers, but this is negligible when compared to the aforementioned.

An estimation of these costs depends on different factors such as the effort put in organizing the benchmarking, the number of tools that participate, the availability of previous experiments that can be reused, and the possibility of automating both the experiment execution and the analysis of the experiment results.

In our case, the costs of organizing the benchmarking were unavoidable, and so were the costs of defining the experiments because no previous experiments that could be reused existed. Furthermore, when this task was carried out the number of participating tools was unknown.

Subject and metrics identification

Once the goals, benefits and costs of the benchmarking have been identified, its scope has to be defined by selecting which Semantic Web tools from the organization will participate, which of its functionalities will be measured, and which evaluation criteria will 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 two benchmarking activities.

As the goal presented in the previous section was too general, the scope was refined to cover a concrete interoperability scenario. Section 1.3.2 presents the different modes that Semantic Web technologies have to interoperate. The most commonly used and, therefore, the one considered here, is the indirect interchange of ontologies by storing them in a shared resource. We have selected this mode because a direct interchange of ontologies would require developing interchange mechanisms for each pair of tools, which would be very costly.

In our case, the shared resource is a local filesystem where ontologies are stored in text files serialized using the RDF/XML syntax because this is the syntax most widely used in Semantic Web technologies.

Also, it was taken into consideration that Semantic Web tools have different knowledge representation models, and it may occur that two tools use the same model or that a tool uses the RDF(S) or the OWL model.

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

The evaluation **metrics** must describe thoroughly the interoperability between an origin tool and a destination one. Therefore, to obtain detailed information on tool interoperability using an interchange language, we need to know

- The components of the knowledge model of an origin tool that can be interchanged with a destination tool¹.
- The secondary effects of interchanging ontologies that include 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 arise when ontologies are interchanged between two tools and the causes of these problems.

¹In the rest of the document, for the sake of clarity, it will appear that “a tool exports/imports/interchanges some components”. This should be understood as “a tool exports/imports/interchanges ontologies that include the realisation of some components”.

Some specific evaluation **criteria** should be established for each experiment to assess the interoperability of the tools. The experiments to be performed should yield data informing how the tools comply with these criteria.

Participant identification

The delimited benchmarking scope helps to identify the organization members that are related to the benchmarking and to form the benchmarking team responsible for continuing the benchmarking activities in the organization.

Because WebODE is being developed by the Ontology Engineering Group at the UPM, it was quite straightforward to identify and contact the members of the organization involved in WebODE's RDF(S) and OWL importers and exporters. In both benchmarking activities, the team was formed by the author and by Jesús Prieto-González, an undergraduate student that provided support by developing experimentation-related software.

Proposal writing

The next task to perform was to compile all the benchmarking-related information into a benchmarking proposal, which should be used as a reference along the benchmarking.

To reach a broader audience, the benchmarking proposals did not take the form of paper documents but of publicly available web pages²³, which include all the relevant information about the benchmarking activities. Currently, this information contains the following:

- Motivation and goals.
- Benefits and costs.
- Tools and people involved.
- Description of the experiment.
- Benchmark suites.
- Planning.
- Related events.
- Results and recommendations.

Management involvement

These benchmarking proposals were presented to the Director of the Ontology Engineering Group and, after her analysis, she agreed on continuing the benchmarking activities and allocating future resources both for performing the experiment and for improving the tool.

²http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

³http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/

Benchmarking partners

Participation in the benchmarking was open to any organization irrespective of being a Knowledge Web partner or not. To find other best-in-class organizations willing to participate, the following actions were taken in the two benchmarking activities:

- To research different ontology development tools, both freely available and commercial ones, which could export and import to and from RDF(S) or OWL and then, to contact the organizations 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 4.2 presents the ontology development tools capable of importing and exporting RDF(S) found by the time of performing this task in the RDF(S) Interoperability Benchmarking (by March 2005). Table 4.3 presents the ontology development tools capable of importing and exporting OWL, which were found by the time of performing this task in the OWL Interoperability Benchmarking (by April 2007). Their developers were directly contacted.

Any Semantic Web tool capable of importing and exporting RDF(S) or OWL could participate in the RDF(S) Interoperability Benchmarking or in the OWL Interoperability Benchmarking, respectively. In the case of the RDF(S) Interoperability Benchmarking, not only ontology development tools, but also RDF repositories participated.

Table 4.4 shows the six tools that took part in the RDF(S) Interoperability Benchmarking, three of which are ontology development tools: KAON, Protégé (using its RDF backend), and WebODE; the other three are RDF repositories: Corese⁴, Jena⁵ and Sesame⁶.

Table 4.5 shows the nine tools having taken part in the OWL Interoperability Benchmarking: one ontology-based annotation tool, namely, GATE⁷; three ontology repositories: Jena⁸, KAON2⁹, and SWI-Prolog¹⁰; and five ontology development tools: the NeOn toolkit¹¹, Protégé-Frames¹², Protégé-OWL¹³, Sementalk¹⁴, and WebODE¹⁵.

⁴<http://www-sop.inria.fr/acacia/soft/corese/>

⁵<http://jena.sourceforge.net/>

⁶<http://www.openrdf.org/>

⁷<http://gate.ac.uk/>

⁸<http://jena.sourceforge.net/>

⁹<http://kaon2.semanticweb.org/>

¹⁰<http://www.swi-prolog.org/packages/semweb.html>

¹¹<http://www.neon-toolkit.org/>

¹²<http://protege.stanford.edu/>

¹³<http://protege.stanford.edu/overview/protege-owl.html>

¹⁴<http://www.semtalk.com/>

¹⁵<http://webode.dia.fi.upm.es/>

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

Table 4.2: Ontology development tools able to import/export RDF(S) by March 2005.

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

Table 4.3: Ontology development tools able to import/export OWL by April 2007.

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

Table 4.4: Tools participating in the RDF(S) Interoperability Benchmarking.

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

Table 4.5: Tools participating in the OWL Interoperability Benchmarking.

As tables 4.4 and 4.5 show, the experiment was not always performed by tool developers. Furthermore, in the RDF(S) Interoperability Benchmarking some tools executed the experiments more times than others because the tools entered the benchmarking at different times (see section 5.2).

In the two benchmarking activities, the tools participating presented a variety of knowledge models. Next, an enumeration of the knowledge models of the tools is presented:

- Corese’s knowledge model enables processing RDF(S) and OWL Lite within the Conceptual Graphs formalism [Corby and Faron-Zucker, 2002].
- GATE’s knowledge model consists of a class hierarchy with a growing level of expressivity. The expressivity of this model is aimed at being broadly equivalent to OWL Lite [Bontcheva *et al.*, 2004].
- Jena’s knowledge model supports RDF and ontology formalisms built on top of RDF. Specifically this means RDF(S), the varieties of OWL, and the now-obsolete DAML+OIL [McBride, 2001].
- KAON’s knowledge model is an extension of RDF(S) that contains the essential modelling primitives of frame-based systems [Motik *et al.*, 2002].

- KAON2's knowledge model is capable of manipulating the $\mathcal{SHIQ}(\mathbf{D})$ subset of OWL-DL and F-Logic [Motik and Sattler, 2006].
- The NeOn Toolkit fully supports F-Logic modelling. Native support of OWL is currently under development [Erdmann and Wenke, 2007].
- Protégé's knowledge model is based on a flexible metamodel, which is comparable to object-oriented and frame-based systems [Noy *et al.*, 2000].
- Protégé-OWL's knowledge model supports RDF(S), OWL Lite, OWL DL and significant parts of OWL Full [Knublauch *et al.*, 2004].
- SemTalk's knowledge model supports modelling RDF(S) and OWL using Visio [Fillies and Weichhardt, 2005].
- Sesame's knowledge model allows managing RDF(S) [Broekstra *et al.*, 2002].
- SWI-Prolog's knowledge model supports RDF(S) and OWL on top of Prolog [Wielemaker *et al.*, 2008].
- WebODE's knowledge model is based in frames and is extracted from the intermediate representations of METHONTOLOGY [Arpírez *et al.*, 2003].

The experiments carried out over the NeOn Toolkit were done in the scope of the NeOn European project¹⁶ and the analysis of the NeOn Toolkit interoperability is presented in [García-Castro, 2007b]. The results of this interoperability are not included in this thesis as they are restricted to the NeOn partners.

Planning and resource allocation

The main deadline of the benchmarking was imposed by that of the benchmarking in Knowledge Web. Therefore, a plan had to be designed that included the *Plan* and *Experiment* phases, though this only contained the first task of the *Improvement* phase (*Benchmarking report writing*).

This plan was developed and agreed on by all the organizations participating in the benchmarking activities; besides, every organization had to assign a number of people to perform the process.

4.4.2. Experiment phase

Experiment definition

The experiments performed in the benchmarking activities had to provide data informing how the Semantic Web tools comply with the evaluation criteria established in the previous phase.

On the other hand, 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)

¹⁶<http://www.neon-project.org/>

and to export ontologies to the language (to write into a file an ontology stored in the tool knowledge model). Therefore, the experiments provided data not only of the interoperability but also of the tool importers and exporters.

As mentioned before, participation in the two benchmarking activities was open to any Semantic Web tool. However, the experiments required that the tools be able to import and export RDF(S) ontologies in one case, and OWL ontologies in the other.

For the experiments, any group of ontologies can be used as input, but having real, large or complex ontologies is useless if we do not know whether the tools can interchange simple ontologies correctly. However, 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, to obtain the required experiment data, the author defined four benchmark suites to be used, which were common to all the tools. Three benchmark suites were used in the RDF(S) Interoperability Benchmarking (the RDF(S) Import, Export, and Interoperability Benchmark Suites) and one in the OWL Interoperability Benchmarking (the OWL Lite Import Benchmark Suite).

The quality of the benchmark suites used is essential for the results of the benchmarking. Therefore, once the benchmark suites were defined, the first step in the two benchmarking activities was to validate these benchmark suites and to agree on their definition. To this end, they were published on the benchmarking web pages so that they could be reviewed by the participants.

The benchmark suites were also validated and refined in reviews performed by Knowledge Web partners in several meetings. In the case of the RDF(S) Interoperability Benchmarking, a workshop was organized by the author in Madrid in October 10th-11th 2005¹⁷, where the participants presented some of their first experiences in using the benchmark suites and evaluated their tools in a hands-on session.

The **experimentation planning** of the two benchmarking activities was defined so as their deadlines would coincide with the Knowledge Web deadlines, date when the benchmarking results should be delivered. Therefore, the planning included the *Plan* and *Experiment* phases, though it contained only the first task of the *Improvement* phase (*Benchmarking report writing*).

This planning was developed and agreed on by all the organizations participating in the benchmarking activities; besides, every organization had to assign a number of people to participate in the experiments.

¹⁷http://knowledgeweb.semanticweb.org/benchmarking_interoperability/working_days/

RDF(S) Interoperability Benchmarking

This chapter presents the instantiation of the benchmarking methodology for the tasks of the *Experiment* phase of the RDF(S) Interoperability Benchmarking. It contains a complete description of the manual approach of the UPM-FBI and of the benchmark suites used in the experiments, namely, the RDF(S) Import, Export and Interoperability Benchmark Suites (figure 5.1); it also provides information about how the experiments were manually performed over the tools participating in the benchmarking, i.e., Corese (version 2.1.2), Jena (version 2.3), KAON (version 1.2.9), Protégé-Frames using its RDF backend (version 3.2 beta build 230), Sesame (version 2.0 alpha 3), and WebODE (version 2.0 build 109); finally, it offers a detailed analysis of the RDF(S) interoperability results.

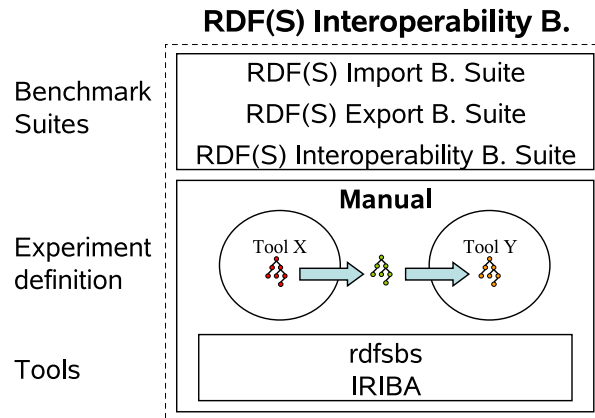


Figure 5.1: UPM-FBI resources for benchmarking RDF(S) interoperability.

The chapter is structured as follows: section 5.1 presents the definition of

the experiment and the benchmark suites used. Section 5.2 describes how and when the experiments were performed. Finally, sections 5.3, 5.4 and 5.5 show, respectively, the results of performing the import, export and interoperability experiments over the different tools.

5.1. Experiment definition

The experiments to be performed must provide data informing how the ontology development tools comply with the evaluation criteria defined in the previous chapter, which are the following:

- The components of the knowledge model of an origin tool that can be interchanged with a destination tool.
- The secondary effects of interchanging ontologies that include 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 arise when ontologies are interchanged between two tools and the causes of these problems.

The interoperability of ontology development tools using an interchange language depends on the capabilities of such tools to import and export ontologies from/to this language. Therefore, as figure 5.2 shows, the experiments included not only an interoperability evaluation but also a previous evaluation of the RDF(S) importers and exporters. Thus, the RDF(S) importer and exporter evaluation results can help to interpret the interoperability results.

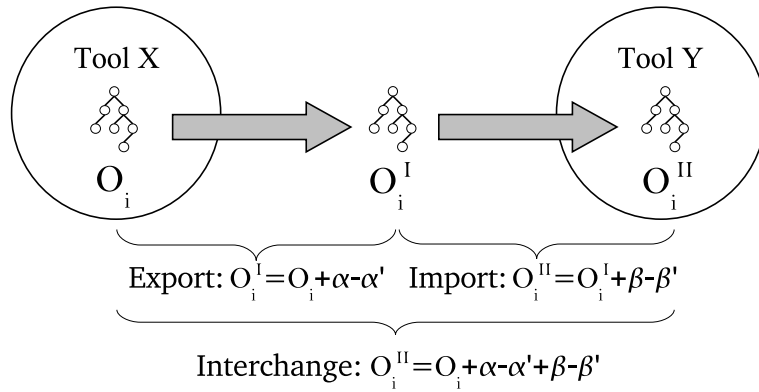


Figure 5.2: Evaluations performed in the RDF(S) experiments.

To obtain the required experiment data, the author has defined three benchmark suites for evaluating the import, export and interoperability capabilities of the tools, which were common for all the tools.

In this setting, it is recommended to perform the import and export experiments before the interoperability ones, because tool interoperability highly depends on the functioning of their importers and exporters and because the effort required to execute the interoperability experiments diminishes when the files produced in the export experiments are used.

This section deals with the definition of these three benchmark suites, which check the correct import, export and interchange of ontologies in the tools participating in benchmarking. It includes, for each of the benchmark suites, the following:

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

5.1.1.1. RDF(S) Import Benchmark Suite

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

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

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

Benchmarks that depend on the knowledge model

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

In order to make the benchmark suite exhaustive, importing all the possible combinations of the knowledge model components was contemplated. Three different types of benchmarks depend on the knowledge model and these are

- Benchmarks that import ontologies with single components (e.g., one *rdfs:Class*).
- Benchmarks that import ontologies with all the possible combinations of two components with a property (e.g., combinations of *rdfs:Class* with the *rdfs:subClassOf* property).
- Benchmarks that import ontologies combining more than two components that usually appear together in RDF(S) graphs (e.g., combinations of *rdf:Property* with *rdfs:domain* and *rdfs:range*).

Appendix A presents the method followed for identifying the different combinations of components of the RDF(S) knowledge model.

As RDF(S) does not impose any restriction for combining its components, the number of the resulting benchmarks is huge (more than 4.000); therefore, the benchmark suite has to be pruned according to its intended use and to the kind of tools that it is supposed to evaluate, namely, ontology development tools.

In this thesis, only the RDF(S) components that have an equivalent in the knowledge models of KAON, Protégé-Frames and WebODE were taken into account: *rdfs:Class*, *rdf:Property*, *rdfs:Literal*, *rdf:type*, *rdfs:domain*, *rdfs:range*, *rdfs:subClassOf*, and *rdfs:subPropertyOf*. The other RDF(S) components were disregarded.

The benchmarks obtained are classified into the following nine groups:

- Class benchmarks
- Metaclass benchmarks
- Subclass benchmarks
- Class and property benchmarks
- Single property benchmarks
- Subproperty benchmarks
- Property with domain and range benchmarks
- Instance benchmarks
- Instance and property benchmarks

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

Benchmarks that depend on the RDF/XML syntax

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

Fifteen benchmarks were defined to take into account the following variants of the RDF/XML syntax:

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

Benchmark definitions

The RDF(S) Import Benchmark Suite contains 82 benchmarks and detailed descriptions of them can be found in appendix B and in the web page¹. Table 5.1 shows the 10 groups of the RDF(S) Import Benchmark Suite, the number of benchmarks in each group, and the RDF(S) components used in each group. All the RDF(S) files to be imported can be downloaded from a single file², whereas templates are provided for collecting the execution results³.

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

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

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

Group	No.	Components used
Class	2	<i>rdfs:Class</i>
Metaclass	5	<i>rdfs:Class</i> , <i>rdf:type</i>
Subclass	5	<i>rdfs:Class</i> , <i>rdfs:subClassOf</i>
Class and property	6	<i>rdfs:Class</i> , <i>rdf:Property</i> , <i>rdfs:Literal</i>
Property	2	<i>rdf:Property</i>
Subproperty	5	<i>rdf:Property</i> , <i>rdfs:subPropertyOf</i>
Property with domain and range	24	<i>rdfs:Class</i> , <i>rdf:Property</i> , <i>rdfs:Literal</i> , <i>rdfs:domain</i> , <i>rdfs:range</i>
Instance	4	<i>rdfs:Class</i> , <i>rdf:type</i>
Instance and property	14	<i>rdfs:Class</i> , <i>rdf:type</i> , <i>rdf:Property</i> , <i>rdfs:Literal</i>
Syntax and abbreviation	15	<i>rdfs:Class</i> , <i>rdf:type</i> , <i>rdf:Property</i> , <i>rdfs:Literal</i>
TOTAL	82	<i>rdfs:Class</i> , <i>rdf:type</i> , <i>rdfs:subClassOf</i> , <i>rdf:Property</i> , <i>rdfs:domain</i> , <i>rdfs:range</i> , <i>rdfs:subPropertyOf</i> , <i>rdfs:Literal</i>

Table 5.1: Groups of the RDF(S) import benchmarks.

The definition of each benchmark in the benchmark suite, as table 5.2 shows, includes the following fields:

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

Evaluation criteria

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

- **Modelling** (YES/NO). The ontology development tool can model the ontology components described in the benchmark.
- **Execution** (OK/FAIL). The execution of the benchmark is normally carried out without any problem, and the tool always produces its expected result. But when an execution fails, the following information is required:
 - Reasons for the benchmark execution failure.
 - If the tool was fixed to pass a benchmark, which corrections the tool required.

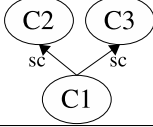
Identifier	I09
Description	Import one class that is subclass of several classes
Graphical representation	
RDF/XML file	<pre> <rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"> <Class rdf:about="http://www.nothing.org/graph09#C1"> <subClassOf rdf:resource="http://www.nothing.org/graph09#C2"/> <subClassOf rdf:resource="http://www.nothing.org/graph09#C3"/> </Class> <Class rdf:about="http://www.nothing.org/graph09#C2"> </Class> <Class rdf:about="http://www.nothing.org/graph09#C3"> </Class> </rdf:RDF> </pre>

Table 5.2: An example of an RDF(S) import benchmark definition.

- **Information added or lost.** It refers to the information added or lost during the ontology interchange when the benchmark is executed.

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

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

The possible combinations of the *Modelling* and *Execution* results are the following:

- **It models and executes.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out without problems, producing the expected result (*Modelling=YES* and *Execution=OK*).
- **It does not model and executes.** The tool does not model the ontology components described in the benchmark and the execution of the benchmark is carried out without problems, producing the expected result (*Modelling=NO* and *Execution=OK*).

- **It models and fails.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out with some problems, or it does not produce the expected result (*Modelling=YES* and *Execution=FAIL*).
- **It does not model and fails.** The tool does not model the ontology components described in the benchmark and the execution of the benchmark is carried out with some problems, or it does not produce the expected result (*Modelling=NO* and *Execution=FAIL*).

Table 5.3 shows an example of executing the benchmark I46 (*Import just one property that has a class as domain and the XML Schema datatype “string” as range, with the class defined in the ontology*) in five fictitious ontology development tools identified as A, B, C, D, and E.

Tool	Modelling	Execution	Information added	Information lost
A	YES	OK	A label in all the components	-
B	YES	FAIL	-	The property’s range
C	NO	OK	The range <i>String</i>	The range <i>xsd:string</i>
D	NO	OK	The range <i>rdfs:Literal</i>	The range <i>xsd:string</i>
E	NO	FAIL	-	The property

Table 5.3: Fictitious results of executing the benchmark I46.

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

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

Because tools C, D and E cannot model the XML Schema datatype *string* as range, though they can model string ranges, the expected result of these tools is to have a property whose domain is a class and whose range is *string*. Tools C and D produce the expected result and their *Execution* result is *OK*; both tools lose information about the range being the XML Schema datatype *string*, though tool C creates the range as its own datatype *String* and tool D creates

the range as *rdfs:Literal*; therefore, these tools, C and D, insert new information in the ontology. Finally, tool E does not import the property at all, although its expected result is to import the property with a string range. The *Execution* result of tool E is *FAIL* and E loses all the information about the property when it imports the ontology.

Procedure for executing the benchmark suite

If a tool developer wants to evaluate the RDF(S) importer of his tool, the steps (shown in figure 5.3) he should follow for executing each benchmark are

1. To specify the result expected from importing the file with the RDF(S) ontology into the ontology development tool, either by modelling the expected ontology in the tool or by defining the ontology informally (e.g. in natural language).
2. To import into the ontology development tool the RDF(S) file that contains the RDF(S) ontology defined in the benchmark.
3. To compare the imported ontology with the expected ontology specified in the first step and to check whether there is some addition or loss of information.

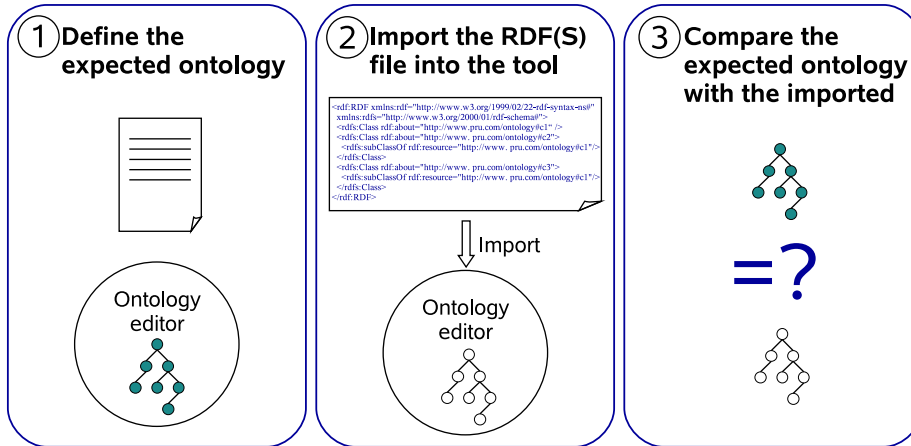


Figure 5.3: Procedure for executing an RDF(S) import benchmark.

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

5.1.2. RDF(S) Export Benchmark Suite

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

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

As in the case of the RDF(S) Import Benchmark Suite, two types of benchmarks were defined for isolating the two issues that influence the correct exporting of an ontology, namely, which combinations of components of the ontology development tool knowledge model are present, and which restrictions are imposed by RDF(S) for naming components. Both types of benchmarks are described in the following sections.

Benchmarks that depend on the knowledge model

These benchmarks check the correct export to RDF(S) of ontologies that model simple combinations of components of a common knowledge model.

To determine this common knowledge model, the knowledge models of KAON, Protégé-Frames, WebODE, and RDF(S) were analysed, and the subset of common components in these models was identified.

This subset of common components has less expressivity than the knowledge models of the tools, and contains the following components: classes and class hierarchies, object and datatype properties, instances, and literals; other components that are specific to each tool were disregarded.

The composition of the RDF(S) Export Benchmark Suite is similar to the composition of the Import one, but instead of taking as input the knowledge model of RDF(S), the common core of knowledge modelling components mentioned above was taken and, therefore, a different number of benchmarks was obtained. The benchmarks obtained are classified into the following ten groups:

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

- Instance and object property benchmarks
- Instance and datatype property benchmarks

Benchmarks that depend on the component naming restrictions

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

- Concepts and properties whose names start with a character other than a letter or `'_'`.
- Concepts and properties with spaces in their names.
- Concepts and properties with URI reserved characters in their names (`'/'`, `'?'`, `':'`, `'@'`, `'&'`, `'='`, `'+'`, `'$'`, `'.'`).
- Concepts and properties with XML delimiter characters in their names (`'<'`, `'>'`, `'#'`, `'%'`, `'\"'`).

Benchmark definitions

Table 5.4 shows the groups of the RDF(S) Export Benchmark Suite, which comprises 66 benchmarks. The table contains the number of benchmarks and the components used in each group. A detailed description of such benchmarks can be found in appendix B and in the web page⁴. In addition, templates are provided for collecting the execution results⁵.

The definition of each benchmark, as table 5.5 shows, includes the following fields:

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

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

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

Group	No.	Components used
Class	2	<i>class</i>
Metaclass	5	<i>class, instanceOf</i>
Subclass	5	<i>class, subClassOf</i>
Class and object property	4	<i>class, object property</i>
Class and datatype property	2	<i>class, datatype property, literal</i>
Object property	14	<i>object property</i>
Datatype property	12	<i>datatype property</i>
Instance	4	<i>class, instanceOf</i>
Instance and object property	9	<i>class, instanceOf, object property</i>
Instance and datatype property	5	<i>class, instanceOf, datatype property, literal</i>
URI character restrictions	4	<i>class, instanceOf, object property, datatype property, literal</i>
TOTAL	66	<i>class, instanceOf, subClassOf, object property, datatype property, literal</i>

Table 5.4: Groups of the RDF(S) export benchmarks.

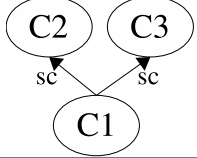
Identifier	E09
Description	Export one class that is subclass of several classes
Graphical representation	 <pre> graph BT C1((C1)) -- sc --> C2((C2)) C1 -- sc --> C3((C3)) </pre>
WebODE's instantiation	Export one concept that is subclass of several concepts
Protégé-Frames' instantiation	Export one class that is subclass of several classes
...	...

Table 5.5: An example of an RDF(S) export benchmark definition.

Evaluation criteria

The evaluation criteria adopted for the export benchmark suite are the same as those for the import benchmark suite, namely, *Modelling*, *Execution* and *Information added or lost*. The only difference with the import criteria is that if a benchmark defines an ontology that cannot be modelled in a certain tool, such benchmark cannot be executed in the tool, being the *Execution* result *N.E.* (Non Executed).

As in the case of the import benchmarks, any combination of results can

be possible since results depend on the decisions taken by tool developers. As mentioned in section 1.3.2, to avoid the loss of knowledge when exporting to a less expressive knowledge model, tools either represent the knowledge that could be lost by using annotation properties, or extend the RDF(S) vocabulary with ad-hoc properties.

The possible combinations of the *Modelling* and *Execution* results are the following:

- **It models and executes.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out without problems, producing the expected result (*Modelling=YES* and *Execution=OK*).
- **It models and fails.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out with some problems or it does not produce the expected result (*Modelling=YES* and *Execution=FAIL*).
- **It does not model.** The tool does not model the ontology components described in the benchmark and, therefore, the benchmark cannot be executed (*Modelling=NO* and *Execution=Non Executed*).

Procedure for executing the benchmark suite

The steps to follow for executing each of the benchmarks, shown in figure 5.4, are

1. To specify the expected ontology that results from exporting the ontology, either in RDF(S) or by defining it informally (e.g. in natural language).
2. To model in the tool the ontology described in the benchmark.
3. To export the ontology modelled using the tool to RDF(S).
4. To compare the exported RDF(S) ontology with the expected RDF(S) ontology specified in the first step, examining whether there is some addition or loss of information.

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

5.1.3. RDF(S) Interoperability Benchmark Suite

The RDF(S) Interoperability Benchmark Suite is used to evaluate the interoperability of Semantic Web tools using RDF(S) as the interchange language. This evaluation is performed by testing the interchange of ontologies from a

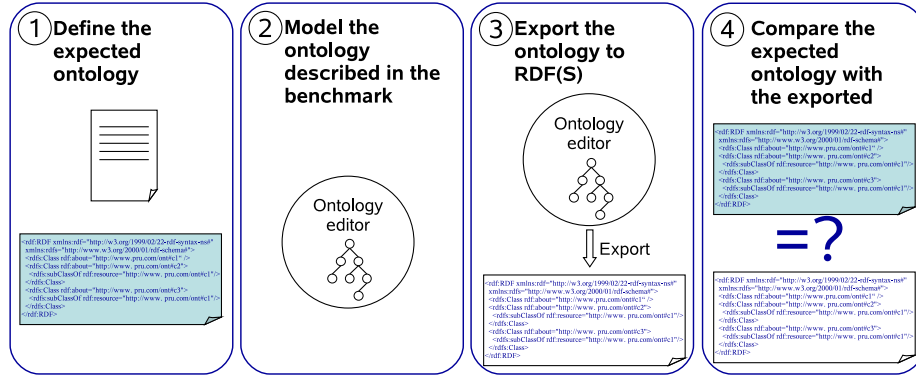


Figure 5.4: Procedure for executing an RDF(S) export benchmark.

source tool to a destination one. Although it was developed for ontology development tools, the benchmark suite can be used to evaluate any other tool capable of importing from and exporting to RDF(S).

The RDF(S) Interoperability Benchmark Suite is composed of benchmarks that check the correct interchange of ontologies between two tools. The benchmark suite considers the interchange of a common subset of the knowledge models of KAON, Protégé-Frames, WebODE, and RDF(S), which is composed of classes and class hierarchies, object and datatype properties, instances, and literals. As these components are the same as those in the RDF(S) Export Benchmark Suite, the ontologies defined in the RDF(S) Interoperability Benchmark Suite are identical to those of the RDF(S) Export Benchmark Suite, presented in the previous section.

The description of the RDF(S) Interoperability Benchmark Suite can be found in appendix B and in the web page⁶; templates are also provided for collecting the execution results⁷. When evaluating the interoperability of the tools that have already executed the RDF(S) Export Benchmark Suite, a file containing the RDF(S) files exported by these tools can be downloaded⁸ and thus these files can be directly imported into the destination tool.

Evaluation criteria

The evaluation criteria adopted here are the same as those adopted for the export benchmark suite, namely, *Modelling*, *Execution* and *Information added or lost*, and as in the export case, any combination of results is possible.

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

⁷http://knowledgeweb.semanticweb.org/benchmarking_interoperability/templates/Interoperability%20Templates.zip

⁸http://knowledgeweb.semanticweb.org/benchmarking_interoperability/stage_1_results/RDFS%20Exported%20Files.zip

The possible combinations of the *Modelling* and *Execution* results are the following:

- **It models and executes.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out without problems, producing the expected result (*Modelling=YES* and *Execution=OK*).
- **It does not model and executes.** The tool does not model the ontology components described in the benchmark and the execution of the benchmark is carried out without problems, producing the expected result (*Modelling=NO* and *Execution=OK*).
- **It models and fails.** The tool models the ontology components described in the benchmark and the execution of the benchmark is carried out with some problems or it does not produce the expected result (*Modelling=YES* and *Execution=FAIL*).
- **It does not model and fails.** The tool does not model the ontology components described in the benchmark and the execution of the benchmark is carried out with some problem or it does not produce the expected result (*Modelling=NO* and *Execution=FAIL*).
- **Not executed.** The origin tool does not model the ontology components described in the benchmark and, therefore, the tool cannot be the origin of the interchange and the benchmark is not executed (*Execution=Non Executed*).

Procedure for executing the benchmark suite

The steps to follow when executing each of the benchmarks (shown in figure 5.5) are

1. To specify the expected ontology resulting from interchanging the ontology in the destination tool, either by modelling the expected ontology in the destination tool or by defining it informally (e.g. in natural language).
2. To model the ontology described in the benchmark in the source tool.
3. To export the ontology modelled using the source tool to an RDF(S) file.
4. To import the RDF(S) file (exported by the source tool) into the destination tool.
5. To compare the interchanged ontology with the expected one specified in the first step, checking whether there is some addition or loss of information.

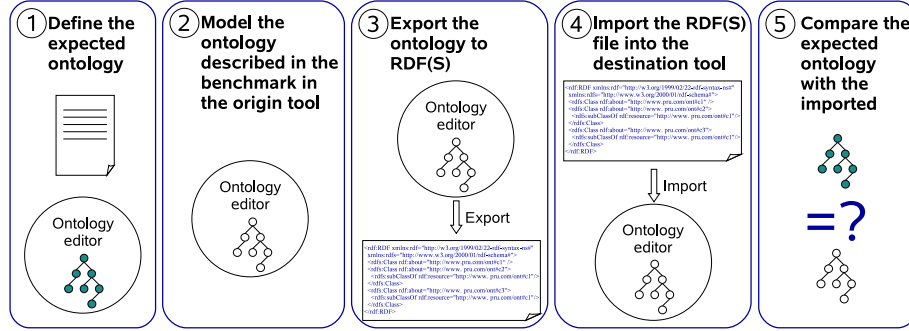


Figure 5.5: Procedure for executing an RDF(S) interoperability benchmark.

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

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

5.2. Experiment execution

The RDF(S) importers and exporters of the ontology development tools were first evaluated with the agreed versions of the benchmark suites. Once the RDF(S) importers and exporters had been evaluated, the evaluation of the interoperability between the ontology development tools was performed.

The evaluation of the RDF(S) importers and exporters of the tools took place in different moments during the development of the import and export benchmark suites. In each new version of the benchmark suites, the coverage of the knowledge models (of RDF(S) and of the tools, respectively) was increased by including new components of the knowledge models and new combinations of the components already included in the benchmark suites.

Not all the tools executed the experiments with all the versions of the benchmark suites because each tool entered the benchmarking at different times. The dates when the benchmark suites were executed and the tools that executed them are the following:

- *April 2005.* WebODE executed a first version of the import and export benchmark suites.

- *May 2005.* Protégé-Frames and WebODE executed a second version of the import and export benchmark suites.
- *October 2005.* KAON, Protégé-Frames, and WebODE executed the third version of the import and export benchmark suites that were sent to the Knowledge Web partners for reviewing them.
- *January 2006.* Corese, Jena, KAON, Protégé-Frames, Sesame, and WebODE executed the final version of the import and export benchmark suites.

5.2.1. Experiments performed

The actual experiments performed over the participating tools are the following:

- Import, export and interoperability experiments were carried out on KAON, Protégé-Frames and WebODE.
- Import and export experiments were carried out on Corese, Jena and Sesame, the RDF repositories.

Benchmarking partners were asked to provide an analysis of the results of their tools, identifying the components that their tools could import, export and interchange as well as the problems encountered. The author compiled all these analyses and provided a general interpretation of them, shown in sections 5.3, 5.4 and 5.5.

5.2.2. Experiment automation

The benchmark suites were intended to be executed manually but, as they contained many benchmarks, it was highly recommended to execute them (or part of them) automatically. In the cases of Corese, Jena, Sesame, and WebODE, most of the experiment was automated. In the other cases, it was performed manually.

The author developed the *rdfsbs* Java application to diminish the effort needed for executing the benchmark suites over WebODE and to provide an easy execution of these benchmark suites. This application allows performing automatically most of the benchmarking experiment in WebODE.

The *rdfsbs* Java application was later adapted by Jesús Prieto González to execute the benchmark suites over Jena and Sesame.

5.3. RDF(S) import results

This section presents the results of executing the common RDF(S) Import Benchmark Suite in all the tools participating in the benchmarking. First, an analysis of the RDF(S) import capabilities of the tools from an in-depth study

of each participating tool is provided. Second, as the tools were evaluated in different moments with different versions of the RDF(S) Import Benchmark Suite, an analysis of the effect of increasing over the time the number of benchmarks in the benchmark suite is performed. And, third, an analysis of the RDF(S) import capabilities of the tools from a global viewpoint is presented. A detailed analysis of the RDF(S) import results can be found in [García-Castro *et al.*, 2006].

The analyses distinguish each possible combination of the *Modelling* and *Execution* results defined in section 5.1.1:

- *It models and executes* (*Modelling*=*YES* and *Execution*=*OK*).
- *It does not model and executes* (*Modelling*=*NO* and *Execution*=*OK*).
- *It models and fails* (*Modelling*=*YES* and *Execution*=*FAIL*).
- *It does not model and fails* (*Modelling*=*NO* and *Execution*=*FAIL*).

5.3.1. KAON RDF(S) import results

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

It models and executes. KAON imports correctly components (or combinations of components) of the RDF(S) knowledge model, which are also present in KAON's own knowledge model. These components are

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

It does not model and executes. Although these components are not present in its own knowledge model, KAON imports correctly ontologies that include the following components (or combinations of components) of the RDF(S) knowledge model:

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

- Instances related through undefined properties or through properties whose range is a XML Schema datatype.

It does not model and fails. KAON does not produce the expected result when it imports some components from RDF(S). The components are the following:

- Class hierarchies with cycles. KAON crashes and it does not import anything.

Syntax and abbreviation. With regard to the import of the different variants of the RDF/XML syntax, we can say that KAON

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

5.3.2. Protégé-Frames RDF(S) import results

This section includes the results of evaluating the RDF(S) import capabilities of Protégé-Frames.

It models and executes. Protégé imports correctly the following components (or combinations of components) of the RDF(S) knowledge model, which are also present in Protégé's own knowledge model:

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

It does not model and executes. Protégé imports correctly ontologies that include components (or combinations of components) of the RDF(S) knowledge model, even though they are not present in Protégé's own knowledge model. These components are

- Classes related through properties that are supposed to be defined with a domain and a range of some metaclass of the classes. Protégé imports the property without domain or range, but it does not relate the classes to the property.
- Properties with undefined resources as domain or range. Protégé creates the undefined resource as a class.
- Properties with multiple ranges. Protégé creates the slot with a range of *Any* because multiple ranges in RDF(S) and Protégé have different meanings.
- Properties with *rdfs:Class* as domain. Protégé does not define the domain as it cannot create a template slot in *:STANDARD_CLASS*.
- Properties with a XML Schema datatype as range. It creates the datatype as a class with the *xsd* namespace.
- Instances of undefined resources. Protégé creates the undefined resource as a class.
- Instances related through undefined properties. Protégé creates the property as a template slot without domain and with a range of *Any*, and it does not relate the instances to the property.
- Instances related through properties with XML Schema datatype as range. Protégé creates the datatype as a class with the *xsd* namespace, but it does not consider the value related to the instance through the property to be an instance of the datatype class nor does Protégé import the value.
- *rdfs:Class*. Protégé imports *rdfs:Class* as *:STANDARD_CLASS*.
- *rdfs:Literal*. Protégé imports *rdfs:Literal* as its own *String* datatype.

It models and fails. Protégé does not produce the expected result when it imports from RDF(S) the following components (or combinations of components) of the RDF(S) knowledge model, which are also present in Protégé's own knowledge model:

- Classes that are instances of multiple metaclasses. Protégé imports the class as instance of only one metaclass instead of importing the class as instance of all the metaclasses. This includes the case in which a class appears in the file as instance of a metaclass and *rdfs:Class*.
- Instances of multiple classes. Protégé imports the instance as instance of only one class instead of importing the instance as instance of all the classes.

It does not model and fails. Protégé does not produce the expected result when it imports from RDF(S) some components (or combinations of components) of the RDF(S) knowledge model, although they are not present in Protégé's own knowledge model. The components are the following:

- Class hierarchies with cycles. Protégé crashes when it finds a cycle in a class hierarchy, so it does not import anything.
- Properties with multiple domains. Protégé imports multiple domains; however, in RDF(S) and in Protégé multiple domains have different meanings. Whereas in RDF(S) multiple domains in properties are considered the intersection of all the domains, in Protégé multiple domains in slots are considered as the union of all the domains.
- Property hierarchies with cycles. Protégé crashes when it finds a cycle in a property hierarchy; therefore, it does not import anything.

Syntax and abbreviation. Regarding the import of the different variants of the RDF/XML syntax, Protégé

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

5.3.3. WebODE RDF(S) import results

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

It models and executes. WebODE imports correctly components (or combinations of components) of the RDF(S) knowledge model that are also present in WebODE's own knowledge model. These components are

- Classes and class hierarchies without cycles.
- Properties with a single domain and range, even if the domain and range are the same class.

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

It does not model and executes. When WebODE imports ontologies with components (or combinations of components) of the RDF(S) knowledge model that are not present in WebODE's own knowledge model, it either imports only those components that it can represent (using equivalent components or using workarounds) or it does not import them.

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

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

- *rdfs:Class*. WebODE imports *rdfs:Class* as an imported term.
- *rdfs:Literal*. WebODE imports *rdfs:Literal* as its own datatype *String*.

It models and fails. WebODE does not produce the expected result when it imports from RDF(S) components (or combinations of components) of the RDF(S) knowledge model, which are also present in WebODE's own knowledge model. These components are the following:

- Properties with XML Schema datatype as range. WebODE imports the datatype as an imported term instead of as a XML Schema datatype.
- Properties whose range is a XML Schema datatype and instances with values in the properties. WebODE inserts its own datatype *String* as the type of the instance attribute instead of the XML Schema datatype.

It does not model and fails. WebODE does not produce the expected result when it imports from RDF(S) some components (or combinations of components) of the RDF(S) knowledge model, although they are not present in WebODE's own knowledge model. The components are the following:

- Properties with multiple domains. WebODE creates an anonymous concept as the origin of the relation (or instance attribute) and as a subclass of one of the domain concepts instead of as a subclass of all the domain concepts.
- Properties with multiple ranges. WebODE creates an anonymous concept as the destination of the relation and as a subclass of one of the range concepts instead of as a subclass of all the range concepts.

Syntax and abbreviation. Regarding the import of the different variants of the RDF/XML syntax, WebODE

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

5.3.4. Corese, Jena and Sesame RDF(S) import results

This section includes the results of evaluating the RDF(S) import capabilities of the RDF repositories, namely, Corese, Jena and Sesame.

The procedure for executing the RDF(S) Import Benchmark Suite in the RDF repositories is different from the one proposed in the benchmark suite. This is so because some of the tasks proposed in the benchmark suite require

functionalities, such as modelling the ontology into the tool, which are available in ontology development tools but not in RDF repositories.

The procedure followed in the RDF repositories for each benchmark was the following:

1. To load the file with the RDF(S) ontology into the tool.
2. To export the loaded ontology to a file using a generic query.
3. To compare the expected ontology with the imported one, checking whether there is some addition or loss of information.

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

5.3.5. Evolution of RDF(S) import results

In this section, the effect of increasing over the time the number of benchmarks in the RDF(S) Import Benchmark Suite is analysed.

This analysis is possible because the tools were evaluated in different moments with different versions of the benchmark suite, as stated in section 5.2. In each new version of the benchmark suite, the coverage of the RDF(S) knowledge model was increased by adding new components of the knowledge model and new combinations of the components already included in the benchmark suite.

Since not all the tools have executed the experiments with all the versions of the benchmark suite, in the result analysis only the results of the tools that executed the import benchmark suite at least two times were considered, namely KAON, Protégé-Frames and WebODE.

Next, the data obtained from the execution of the RDF(S) Import Benchmark Suite over the time in KAON (table 5.6 and figure 5.6), Protégé-Frames (table 5.7 and figure 5.7), and WebODE (table 5.8 and figure 5.8) is presented.

The tables show, for each execution of the benchmark suite, the number of benchmarks that the tool models and executes (*Modelling=YES* and *Execution=OK*), the number of benchmarks that the tool models and fails (*Modelling=YES* and *Execution=FAIL*), the number of benchmarks that the tool does not model but executes (*Modelling=NO* and *Execution=OK*), the number of benchmarks that the tool does not model and fails (*Modelling=NO* and *Execution=FAIL*), the number of benchmarks, the number of new benchmarks, the number of new benchmarks that are modelled by the tool (*Modelling=YES*), the percentage of benchmarks modelled by the tool from the new benchmarks, the number of benchmarks failed by the tool (*Execution=FAIL*), the number of new benchmarks failed by the tool, and the percentage of failed benchmarks from the new benchmarks.

The figures mentioned above present the four first rows of the tables (the number of benchmarks that the tool models and executes, models and fails, does not model and executes, and does not model and fails).

	First version	Second version	Third version	Final version
Models and executes	-	-	69	79
Does not model and executes	-	-	0	0
Models and fails	-	-	1	1
Does not model and fails	-	-	2	2
Number of benchmarks	-	-	72	82
New benchmarks	-	-	72	10
New benchmarks modelled	-	-	70	10
% New benchmarks modelled / new benchmarks	-	-	97%	100%
Failed benchmarks	-	-	3	3
New failed benchmarks	-	-	3	0
% Failed benchmarks / new benchmarks	-	-	4%	0%

Table 5.6: Summary of KAON's RDF(S) import result evolution.

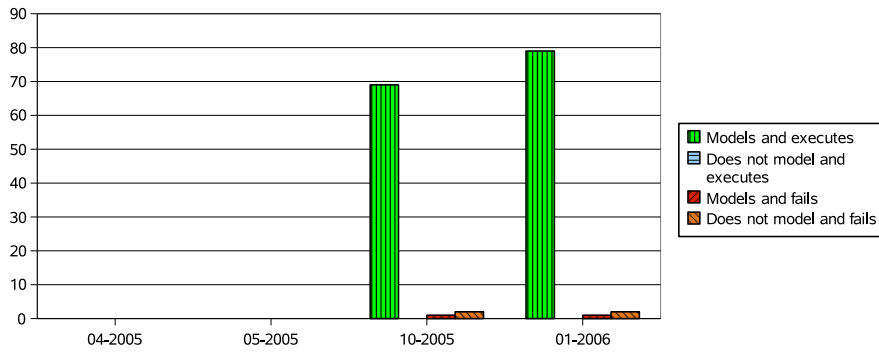


Figure 5.6: Evolution of the RDF(S) import results in KAON.

	First version	Second version	Third version	Final version
Models and executes	-	13	45	48
Does not model and executes	-	5	17	24
Models and fails	-	0	2	2
Does not model and fails	-	2	8	8
Number of benchmarks	-	35	72	82
New benchmarks	-	35	37	10
New benchmarks modelled	-	13	34	3
% New benchmarks modelled / new benchmarks	-	65%	65%	30%
Failed benchmarks	-	2	10	10
New failed benchmarks	-	2	8	0
% Failed benchmarks / new benchmarks	-	10%	15%	0%

Table 5.7: Summary of Protégé-Frames' RDF(S) import result evolution.

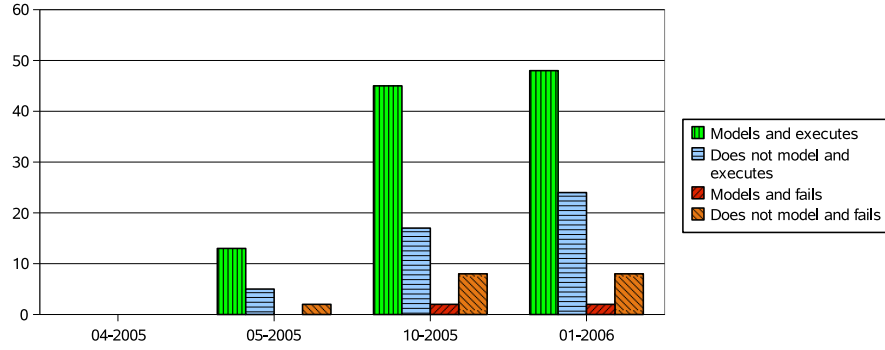


Figure 5.7: Evolution of the RDF(S) import results in Protégé-Frames.

	First version	Second version	Third version	Final version
Models and executes	15	21	39	47
Does not model and executes	3	5	27	25
Models and fails	1	3	6	3
Does not model and fails	2	6	0	7
Number of benchmarks	21	35	72	82
New benchmarks	21	14	37	10
New benchmarks modelled	16	8	21	5
% New benchmarks modelled / new benchmarks	76%	57%	57%	50%
Failed benchmarks	3	9	9	10
New failed benchmarks	3	6	4	3
% Failed benchmarks / new benchmarks	14%	43%	11%	30%

Table 5.8: Summary of WebODE's RDF(S) import result evolution.

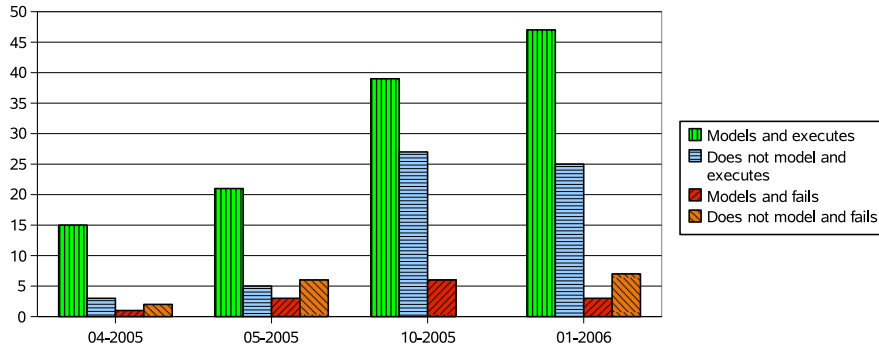


Figure 5.8: Evolution of the RDF(S) import results in WebODE.

The first examination to perform over this data is to **detect whether more software defects appear when the number of benchmarks increases**. To do this, the number of new failed benchmarks, which appears in tables 5.6, 5.7 and 5.8, has been considered.

As for WebODE, every time the number of benchmarks is increased, more benchmarks fail. But in the case of KAON and Protégé-Frames, the number of failed benchmarks does not increase between the last and the penultimate execution.

In general, in our case it cannot be said that an increase in the number of benchmarks has a direct effect on the number of failed benchmarks. The increase depends on the tool and on the kind of benchmarks inserted.

Furthermore, the number of failed benchmarks is not related to the number of defects in the software. One single software defect can affect one or more benchmarks; therefore, increasing the number of benchmarks can cause increasing the number of benchmarks affected by one defect, but not finding new defects.

Failures in benchmarks do not always come from the new benchmarks inserted. In the case of WebODE, its developers corrected some defects but, while doing so, they unintentionally inserted new ones.

We can also **analyse whether the coverage of the benchmarks in the knowledge models of the tools increases when the number of benchmarks increases**. To carry out this analysis, we considered the number of new benchmarks that are modelled by the tool and the ratio of this number to the amount of new benchmarks, which appear in tables 5.6, 5.7 and 5.8.

In every case the coverage of the benchmarks in the knowledge model of the tools increases when the number of benchmarks increases, but the coverage increases at a different rate in each tool (100% in KAON, 30-65% in Protégé-Frames and 50-57% in WebODE).

We can say that the increase in the coverage depends on each tool knowledge model. The more similar a tool knowledge model is to the RDF(S) knowledge model, the greater the coverage when the number of benchmarks is increased.

The results allow us to **analyse if the tools improve over the time**. To do this, we have considered the number of failed benchmarks when their components can be modelled by the tool and when they cannot (see tables 5.6, 5.7 and 5.8).

With regard to Protégé-Frames and KAON, the number of failed benchmarks just increased but with regard to WebODE, the number of failed benchmarks increased and decreased between executions. This decrease implies a commitment from the organization that develops the tool, as changes have been performed to correct defects in the tool.

Generally speaking, we can say that a decrease in the number of failed benchmarks implies that the tool is being improved. And conversely, an increase in the number of failed benchmarks implies either that new benchmarks cause new benchmark failures or that correcting one defect can produce other defects in software.

Finally, after these comments, we can **analyse whether it is worth increasing more the number of benchmarks** in the RDF(S) Import Benchmark Suite. We must bear in mind that in our scenario the execution and analysis of results is a time-consuming task and that a higher number of benchmarks entails a higher cost in executing the benchmark suite.

The effectiveness of this increase depends on the new benchmarks inserted and on the knowledge models of the tools. Below the two scenarios in which the increase is possible are presented:

- To include new combinations of the RDF(S) components currently considered would not be useful because more complex combinations are built from the simple combinations already taken into account.
- To include new components or combinations of components not taken into account would be useful for the tools capable of modelling these components because they have not yet been evaluated.

5.3.6. Global RDF(S) import results

Figure 5.9⁹ presents a quantitative analysis of the global results of the last execution of the RDF(S) Import Benchmark Suite in January 2006. For each tool, it shows the number of benchmarks that fall into each possible combination of the *Modelling* and *Execution* results defined in section 5.1.1.

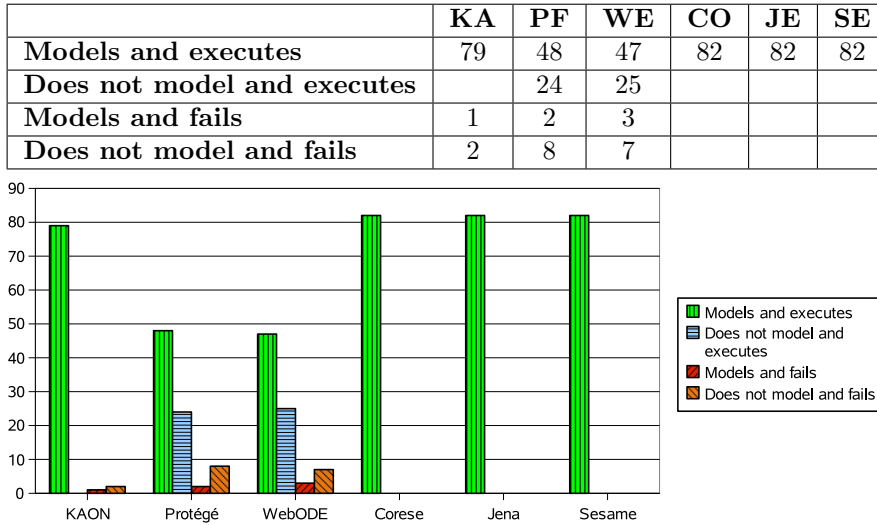


Figure 5.9: Final RDF(S) import results.

⁹The tool names have been abbreviated in the table: KA=KAON, PF=Protégé-Frames, WE=WebODE, CO=Corese, JE=Jena, SE=Sesame

In the benchmarking, the only RDF(S)-native participant tools are Corese, Jena and Sesame, i.e., the RDF repositories. The ontology development tools (KAON, Protégé-Frames and WebODE) have their own knowledge models, and only some of their components can also be represented in RDF(S).

In the figure, we can see that the tools that do not have RDF(S) as their native model have execution problems, both in the components that they are able to model and in those that they are not. On the contrary, the tools with RDF(S) knowledge model execute correctly all the benchmarks.

Therefore, the results obtained when importing from RDF(S) depend mainly on the knowledge model of the tool that executed the benchmark suite. The tools that naturally support the RDF(S) knowledge model (the RDF repositories in our case) do not need to perform any translation in the ontologies when importing them from RDF(S), they import correctly all the combinations of components. As for tools with knowledge models that are different from RDF(S), they do need to translate ontologies from RDF(S) to their knowledge model.

In general, the ontology development tools import correctly from RDF(S) most of the combinations of components that they model, rarely adding or losing information. But when dealing with the tools specifically, they behave as follows:

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

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

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

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

- Import correctly resources with different URI reference syntaxes.

- Import correctly resources with different syntaxes (shortened and unshortened) of empty nodes, multiple properties, typed nodes, string literals, and blank nodes. The only exceptions are 1) KAON, when it imports resources with multiple properties in the unshortened syntax, and 2) Protégé-Frames, when it imports resources with empty and blank nodes in the unshortened syntax.
- Do not import language identification attributes (*xml:lang*) in tags.

If we classify the mismatches found when comparing the imported ontology with the expected one according to the levels of ontology translation problems described in section 1.3.3, we can see that all the mismatches belong to the *Conceptual* level (i.e., they are due to differences in conceptualizations).

5.4. RDF(S) export results

This section presents the results of executing the common RDF(S) Export Benchmark Suite in all the tools participating in the benchmarking. First, an analysis of the RDF(S) export capabilities of the tools from a thorough study of each participating tool is provided. Second, as the tools were evaluated in different moments with different versions of the RDF(S) Export Benchmark Suite, an analysis of the effect of increasing over the time the number of benchmarks in the benchmark suite is performed. And, third, an analysis of the RDF(S) export capabilities of the tools from a global viewpoint is presented. A detailed analysis of the RDF(S) export results can be found in [García-Castro *et al.*, 2006].

The analyses distinguish each possible combination of the *Modelling* and *Execution* results defined in section 5.1.2:

- *It models and executes* (*Modelling=YES* and *Execution=OK*).
- *It models and fails* (*Modelling=YES* and *Execution=FAIL*).
- *It does not model* (*Modelling=NO* and *Execution=Non Executed*).

5.4.1. KAON RDF(S) export results

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

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

- Classes, metaclasses, and class hierarchies without cycles.
- Datatype properties with a domain and a range, without domain and with range, or with multiple domains.

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

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

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

It models and fails. KAON does not produce the expected result when it exports to RDF(S)

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

It does not model. The KAON knowledge model does not include components (or combinations of components) that appear in benchmark definitions and, therefore, KAON cannot export them. These components are

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

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

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

5.4.2. Protégé-Frames RDF(S) export results

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

It models and executes. Protégé exports correctly to RDF(S) components (or combinations of components) of its knowledge model, also present in the RDF(S) knowledge model. Such components are

- Classes, metaclasses, and class hierarchies without cycles.
- Classes and instances that are instances of one class only.
- Template slots with or without domain or range.
- Instances related through template slots, even when the slot relates instances of the same class, or it relates an instance to itself, or when an instance is related through the same slot (either as subject or object) to several other instances.

Protégé exports correctly components (or combinations of components) of its knowledge model, although they are not present in the RDF(S) knowledge model. In these cases, Protégé

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

It models and fails. Protégé does not produce the expected result when it exports to RDF(S)

- Classes or instances that are instances of multiple classes. Protégé only exports the resource as instance of one class instead of exporting the resource as instance of all the classes.
- Template slots with multiple domains. Protégé exports the multiple domains, but multiple domains have different meaning in Protégé and in RDF(S). In Protégé, multiple domains in slots are considered as the union of all the domains, whereas in RDF(S), multiple domains in properties are considered the intersection of all the domains.

It does not model. The Protégé knowledge model does not include some components (or combinations of components) that appear in benchmark definitions and, therefore, Protégé cannot export them. These components are

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

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

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

5.4.3. WebODE RDF(S) export results

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

It models and executes. WebODE produces the expected result in all the benchmarks when it exports to RDF(S).

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

- Concepts and concept hierarchies without cycles.
- Instance attributes of a concept, either with a WebODE datatype or with a XML Schema datatype.

- Relations between two concepts or between a concept and itself.
- Instances of only one concept.
- Instances related through relations, even when the relation is held between instances of the same concept or between an instance with itself, or when an instance has the same relation (either as origin or destination) to several other instances.
- Instances with instance attributes, even when an instance has several values in the instance attribute, or when the instance attribute has an XML Schema datatype as type.

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

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

It does not model. The WebODE knowledge model does not include some components (or combinations of components) that appear in benchmark definitions and, therefore, WebODE cannot export them. These components are

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

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

- Does not modify the name of a concept, of an instance attribute, or of a relation when it does not start with a letter or “_”.
- Encodes spaces in concept names, in instance attribute names, and in relation names as “_”.
- Does not modify the name of a concept, of an instance attribute, or of a relation when it includes URI reserved characters.
- Cannot model concepts and properties with the character ‘”’ in their names.

5.4.4. Corese, Jena and Sesame RDF(S) export results

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

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

The procedure followed in the RDF repositories was the following:

1. To specify the expected ontology that results from exporting the ontology, either in RDF(S) or by defining it informally (e.g. in natural language).
2. To define a fictitious RDF(S) ontology that covered all the combinations of components present in the benchmark suite and to store this ontology into the repository.
3. To define queries for extracting, from the fictitious ontology, ontologies with the combinations of components required in each benchmark. For Corese, Jena and Sesame the queries were defined in the SPARQL query language¹⁰.
4. To export the ontologies with the combinations of components by running the queries against the ontology and saving the results in separate RDF(S) files.
5. To compare the exported RDF(S) ontologies with the expected ones, checking whether there is some addition or loss of information.

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

¹⁰<http://www.w3.org/TR/rdf-sparql-query/>

5.4.5. Evolution of RDF(S) export results

In this section, the effect of increasing over the time the number of benchmarks in the RDF(S) Export Benchmark Suite is analysed.

As stated in section 5.2 above, the tools were evaluated in different moments with different versions of the benchmark suite. In each new version the coverage of the common knowledge model of all the tools was increased by adding new components of this knowledge model and new combinations of the components already included in the benchmark suite.

Since not all the tools have executed the experiments with all the versions of the benchmark suite, in this analysis only the results of the tools that have executed the export benchmark suite at least two times are considered, namely, KAON, Protégé-Frames and WebODE.

Next, the data obtained from the execution of the RDF(S) Export Benchmark Suite over the time in KAON (table 5.9 and figure 5.10), Protégé-Frames (table 5.10 and figure 5.11), and WebODE (table 5.11 and figure 5.12) is presented.

For each execution of the benchmark suite, the tables show the number of benchmarks that the tool models and executes (*Modelling=YES* and *Execution=OK*), the number of benchmarks that the tool models and fails (*Modelling=YES* and *Execution=FAIL*), the number of benchmarks that the tool does not model and therefore does not execute (*Modelling=NO* and *Execution=Non Executed*), the number of benchmarks, the number of new benchmarks, the number of new benchmarks that are modelled by the tool (*Modelling=YES*), the percentage of benchmarks modelled by the tool from the new benchmarks, the number of benchmarks failed by the tool (*Execution=FAIL*), the number of new benchmarks failed by the tool, and the percentage of failed benchmarks from the new benchmarks.

The figures show the three first rows of the tables (the number of benchmarks that the tool models and executes, that models and fails, and that does not model and does not execute).

The first examination we made over this data was to **analyse whether more software defects are detected when the number of benchmarks increases**. To do this, we have to consider the number of new failed benchmarks, which appears in tables 5.9, 5.10 and 5.11.

Sometimes, when KAON, Protégé-Frames and WebODE execute the new benchmarks, they fail. The comments made for the case of the RDF(S) Import Benchmark Suite can also be applied here; that is, an increase in the number of benchmarks does not affect directly the number of failed benchmarks, since this effect depends on the tool and on the kind of benchmark inserted.

We can also **analyse if the coverage of the benchmarks in the knowledge models of the tools increases when the number of benchmarks increases**. To do this, we considered the number of new benchmarks modelled by the tool and the ratio of this number to the number of new benchmarks, which appear in tables 5.9, 5.10 and 5.11.

In every case, the coverage of the benchmarks in the knowledge model of

	First version	Second version	Third version	Final version
Models and executes	-	-	46	54
Models and fails	-	-	2	3
Does not model	-	-	4	9
Number of benchmarks	-	-	52	66
New benchmarks	-	-	52	14
New benchmarks modelled	-	-	48	9
% New benchmarks modelled - new benchmarks	-	-	92%	64%
Failed benchmarks	-	-	2	3
New failed benchmarks	-	-	2	1
% Failed benchmarks - new benchmarks	-	-	4%	7%

Table 5.9: Summary of KAON's RDF(S) export result evolution.

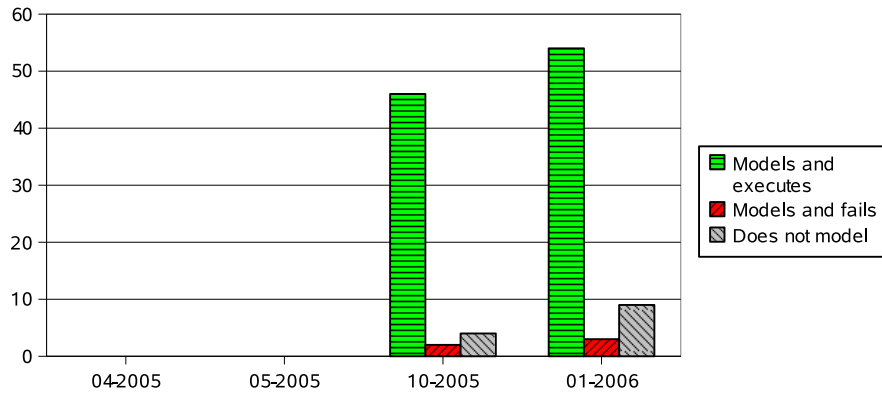


Figure 5.10: Evolution of the RDF(S) export results in KAON.

	First version	Second version	Third version	Final version
Models and executes	-	9	37	40
Models and fails	-	0	7	8
Does not model	-	2	8	18
Number of benchmarks	-	11	52	66
New benchmarks	-	11	41	14
New benchmarks modelled	-	9	35	4
% New benchmarks modelled - new benchmarks	-	82%	85%	29%
Failed benchmarks	-	0	7	8
New failed benchmarks	-	0	7	1
% Failed benchmarks - new benchmarks	-	0%	17%	7%

Table 5.10: Summary of Protégé-Frames' RDF(S) export result evolution.

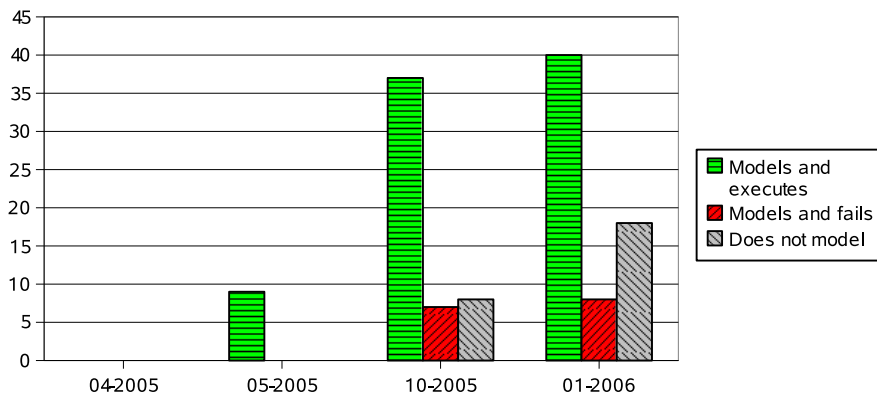


Figure 5.11: Evolution of the RDF(S) export results in Protégé-Frames.

	First version	Second version	Third version	Final version
Models and executes	7	11	18	25
Models and fails	0	0	1	0
Does not model	0	0	33	41
Number of benchmarks	7	11	52	66
New benchmarks	7	4	41	14
New benchmarks modelled	7	4	8	6
% New benchmarks modelled - new benchmarks	100%	100%	20%	43%
Failed benchmarks	0	0	1	0
New failed benchmarks	0	0	1	0
% Failed benchmarks - new benchmarks	0%	0%	2%	0%

Table 5.11: Summary of WebODE's RDF(S) export result evolution.

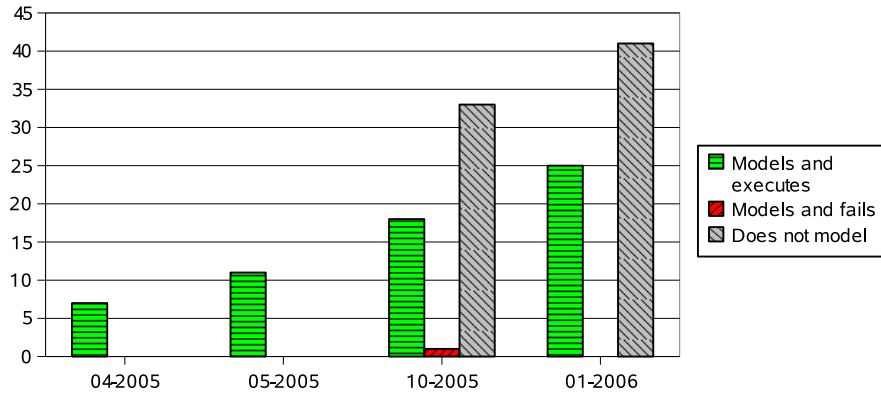


Figure 5.12: Evolution of the RDF(S) export results in WebODE.

the tools increases as the number of benchmarks increases. But, in general, the rate of new benchmarks modelled decreases over the time. This is logical if we bear in mind that we started with a group of components common to the knowledge models of all the tools, but when we extended the benchmark suite we considered new components that are not common to all of the tools.

Besides, the results allowed us to **determine whether the tools improve over the time**. To do this, we considered the number of failed benchmarks, which appear in tables 5.9, 5.10 and 5.11.

With regard to KAON and Protégé-Frames, the number of failed benchmarks just increased, but with regard to WebODE, the number of failed benchmarks increased and decreased between executions. This decrease implies that changes were performed to correct defects in the tool; it also highlights that the results are valid for the time in which the experiments are performed and that these results can change with future releases of the tools.

As mentioned when analysing the import results, a decrease in the number of failed benchmarks implies that the tool is being improved. And an increase in the number of failed benchmarks implies either that new benchmarks cause new benchmark failures or that correcting one defect can produce other defects in software.

Finally, after these comments, we can **analyse if it is worth increasing more the number of benchmarks** in the RDF(S) Export Benchmark Suite. We have to take into account that in our scenario the execution and analysis of results is a time-consuming task and that a higher number of benchmarks entails a higher cost in executing the benchmark suite.

We can say here what we said for the RDF(S) Import Benchmark Suite. The effectiveness of this increase depends on the new benchmarks inserted and on the knowledge models of the tools.

5.4.6. Global RDF(S) export results

Figure 5.13 presents a quantitative analysis of the global results of the last execution of the RDF(S) Export Benchmark Suite in January 2006. For each tool it shows the number of benchmarks that fall into each of the possible combinations of the *Modelling* and *Execution* results defined in section 5.1.2.

In this figure, we can see that only KAON and Protégé-Frames pose some problems when they export ontologies to RDF(S). The rest of the tools export correctly.

Furthermore, the lower the number of benchmarks not modelled is, the more similar the knowledge model of the tool is to the common knowledge model considered, which contains the subset of common components in KAON, Protégé-Frames, WebODE, and RDF(S). In our case, the knowledge models of the RDF repositories are the most similar, followed by those of KAON, Protégé-Frames, and WebODE (in this order).

The number of benchmarks not modelled that appears in the RDF repositories results corresponds with the benchmarks that check the component naming

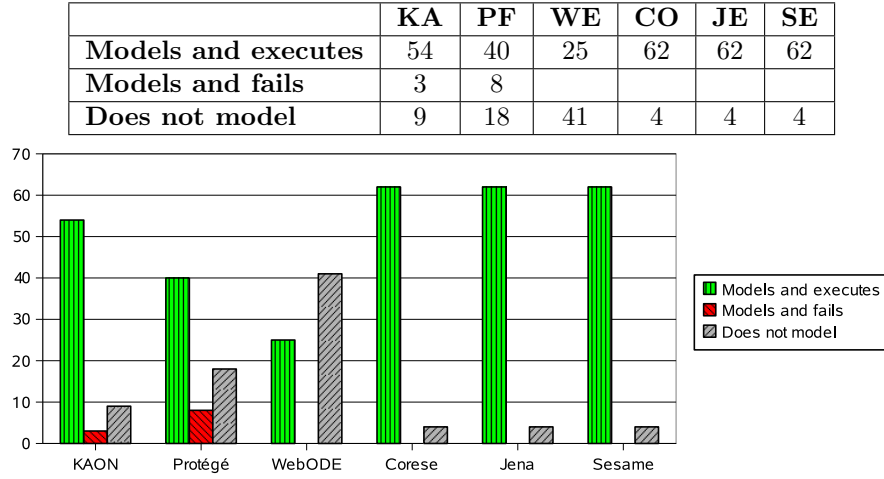


Figure 5.13: Final RDF(S) export results.

restrictions. As these restrictions cannot be modelled in RDF(S), these benchmarks cannot be executed in the RDF repositories.

When we do not consider these benchmarks, we can observe that the common knowledge model is totally compatible with the RDF(S) knowledge model and partially compatible with KAON, Protégé-Frames and WebODE (listed in decreasing compatibility order).

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

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

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

- KAON poses problems only when it exports to RDF(S) datatype properties without range and datatype properties with multiple domains and a XML Schema datatype as range.
- Protégé-Frames poses problems only when it exports to RDF(S) classes or instances that are instances of multiple classes, and template slots with multiple domains.
- WebODE exports correctly to RDF(S) all the combinations of components.

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

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

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

If we classify the mismatches found when comparing the exported ontology with the expected one according to the levels of ontology translation problems described in section 1.3.3, we can see that most of the mismatches belong to the *Conceptual* level (i.e., they are due to differences in conceptualizations). The only exceptions are those mismatches produced at the *Lexical* level by benchmarks that deal with component naming restrictions because in some cases tools do not encode component names in the translation.

5.5. RDF(S) interoperability results

This section presents the results of executing the common RDF(S) Interoperability Benchmark Suite in the ontology development tools participating in the benchmarking. First, a thorough analysis of the interoperability capabilities of each tool is provided. Second, a global analysis of the interoperability of the tools is performed. A detailed analysis of the RDF(S) interoperability results can be found in [García-Castro *et al.*, 2006].

The analyses distinguish each possible combination of the *Modelling* and *Execution* results defined in section 5.1.3: *It models and executes* (*Modelling*=YES and *Execution*=OK), *It does not model and executes* (*Modelling*=NO and *Execution*=OK), *It models and fails* (*Modelling*=YES and *Execution*=FAIL), *It does not model and fails* (*Modelling*=NO and *Execution*=FAIL), and *Not executed* (*Execution*=Non Executed).

5.5.1. KAON interoperability results

Table 5.12 shows the different combinations of components that can be modelled in KAON, classified into categories, and explains whether these components can be interchanged from the other tools to KAON or not. The combinations of components of the tool common knowledge model that cannot be modelled in KAON and, therefore, cannot be interchanged with KAON are not shown in the table. The cells in the table (and in the other tables of this section) include

- **OK** when the *Execution* results of all the benchmarks in the category are *OK*.
- **FAIL** when the *Execution* result of some benchmark in the category is *FAIL*.
- **'-'** when the combination of components in the category cannot be modelled in the source tool.

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

Table 5.12: RDF(S) interoperability results from all the tools to KAON.

It models and executes. The combinations of components that can be interchanged from the tools that can model the combination of components to KAON are

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

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

It models and fails. The other combinations of components are not interchanged, though KAON can model them, for the following reasons:

- Classes that are instances of multiple metaclasses. KAON cannot receive classes that are instances of multiple metaclasses from Protégé, but it can receive them from Corese and from itself.
- Datatype properties without domain and with range. KAON cannot receive datatype properties without domain and with range from Protégé because the information about the range is lost during the interchange, but it can receive them from Corese and from itself.
- Datatype properties whose range is *String*. KAON cannot receive datatype properties whose range is *String* from Protégé, nor from WebODE because the information about the range is lost during the interchange, but it can receive them from Corese and from itself.
- Instances related through datatype properties. KAON cannot receive instances related through datatype properties from Protégé because the information about the range is lost during the interchange, but KAON can receive them from Corese, from WebODE and from itself.

It does not model. Some combinations of components of the tool common knowledge model cannot be modelled in KAON and, therefore, cannot be interchanged with KAON. Nevertheless, the interoperability experiments from the tools that can model these combinations of components to KAON provide some information for these combinations of components:

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

URI character restrictions. With regard to the interchange of classes and properties with URI character restrictions in their names, we can say that KAON

- Interchanges with WebODE and with itself classes and properties whose name starts with a character that is not a letter nor '.', but it does not interchange them with Protégé because the information about the range of the property is not imported.
- Interchanges with WebODE and with itself classes and properties with spaces in their names, but it does not interchange them with Protégé because the information about the range of the property is not imported.
- Interchanges with WebODE and with itself classes and properties with URI reserved characters in their names, but it does not interchange them with Protégé because the information about the range of the property is not imported.
- Interchanges with itself classes and properties with XML delimiter characters in their names, but it does not interchange them with Protégé because the information about the range of the property is not imported.

5.5.2. Protégé-Frames interoperability results

Table 5.13 illustrates the different combinations of components that can be modelled in Protégé-Frames, classified into categories; the table also shows whether these components can be interchanged from the other tools to Protégé-Frames or not. The combinations of components of the tool common knowledge model that cannot be modelled in Protégé-Frames and, therefore, cannot be interchanged with Protégé-Frames are not shown in the table.

It models and executes. The combinations of components that can be interchanged from the tools that can model the combination of components to Protégé are

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

It models and fails. The other combinations of components are not interchanged, even though Protégé can model them, for the following reasons:

- Classes that are instances of a single metaclass. Protégé cannot import the statement of a class being instance of a single metaclass if in the file that class also appears as instance of *rdfs:Class*. This is because Protégé does not import the statement of a class being instance of multiple metaclasses (even if one of these metaclasses is *rdfs:Class*).
- Classes instance of multiple metaclasses. Protégé cannot import the statement of a class being instance of multiple metaclasses. This is because Protégé does not import the statement of a class being instance of multiple metaclasses (even if one of these metaclasses is *rdfs:Class*).

Combination of components	CO	KA	PF	WE
Classes	OK	OK	OK	OK
Classes instance of a single metaclass	FAIL	FAIL	OK	-
Classes instance of a multiple metaclasses	FAIL	FAIL	FAIL	-
Class hierarchies without cycles	OK	OK	OK	OK
Datatype properties without domain and range	FAIL	FAIL	OK	-
Datatype properties with domain but without range	OK	FAIL	OK	-
Datatype properties without domain but with range	FAIL	OK	OK	-
Datatype properties whose range is String	FAIL	OK	OK	OK
Object properties without domain and range	OK	OK	OK	-
Object properties with domain but without range	OK	OK	OK	-
Object properties without domain but with range	OK	OK	OK	-
Object properties with a domain and a range	OK	OK	OK	OK
Instances of a single class	OK	OK	OK	OK
Instances of multiple classes	FAIL	FAIL	FAIL	-
Instances related through object properties	FAIL	OK	OK	OK
Instances related through datatype properties	FAIL	OK	OK	OK

Table 5.13: RDF(S) interoperability results from all the tools to Protégé-Frames.

- Datatype properties without domain or range, or with range *String*. Protégé crashes and does not import anything when a XML Schema datatype (i.e. *xsd:integer*) is defined in the file as an *rdfs:Datatype*.
- Instances of multiple classes. Protégé does not import instances that are instances of multiple classes.
- Instances related through object and datatype properties. Protégé crashes when it imports properties with *rdf:datatype* attributes.

It does not model. Some combinations of components of the tool common knowledge model cannot be modelled in Protégé and, therefore, cannot be interchanged with Protégé. Nevertheless, the interoperability experiments from the tools that can model these combinations of components to Protégé provide some information for these combinations of components:

- Class hierarchies with cycles. When Protégé finds a cycle in a class hierarchy from Corese, it crashes and does not import anything.
- Classes related through object properties. Protégé does not import the

property between the class and another class nor the domain and range of the property.

- Classes related through datatype properties. Protégé does not import the property between the class and a datatype nor the domain and range of the property.
- Object and datatype properties with undefined resources as domain or range. Protégé creates the undefined resource as a class.
- Object and datatype properties with multiple domains. Protégé creates the property as a template slot with multiple domains. This is not the expected result because in Protégé multiple domains in slots are considered the union of all the domains, whereas in RDF(S) multiple domains in properties are considered the intersection of all the domains.
- Object properties with multiple ranges. Protégé creates the property as a template slot with a range of *Any*.
- Datatype properties whose range is a XML Schema datatype. When Protégé receives ontologies from Corese, it crashes and does not import anything when a XML Schema datatype (i.e. *xsd:integer*) is defined in the file as an *rdfs:Datatype*.

When Protégé receives ontologies from KAON and WebODE, it creates the XML Schema datatype as a class, which is the range of the property.

- Instances of undefined resources. Protégé creates the undefined resource as a class.
- Instances related through undefined object and datatype properties. Protégé creates the property as a template slot without a domain and with a range of *Any*. The property between the instances is not created.
- Instances related through datatype properties whose range is a XML Schema datatype. Protégé creates the XML Schema datatype as a class and this class is the range of the property; however, it does not import the literal value of the property.

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

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

- Interchanges with WebODE classes and properties with URI reserved characters in their names, but it does not interchange them with KAON and with itself because when exporting they replace the illegal character with ' _ '.
- Does not interchange with any tool classes and properties having XML delimiter characters in their names.

5.5.3. WebODE interoperability results

Table 5.14 illustrates the different combinations of components that can be modelled in WebODE, classified into categories; the table also shows whether these components can be interchanged from the other tools to WebODE or not. The combinations of components of the tool common knowledge model that cannot be modelled in WebODE and, therefore, cannot be interchanged with WebODE are not shown in the table.

Combination of components	CO	KA	PF	WE
Classes	OK	OK	OK	OK
Class hierarchies without cycles	FAIL	OK	OK	OK
Datatype properties with domain and whose range is <i>String</i>	FAIL	OK	OK	OK
Datatype properties with domain and whose range is a XML Schema datatype	OK	OK	-	OK
Object properties with a domain and a range	OK	OK	OK	OK
Instances of a single class	OK	OK	OK	OK
Instances related through object properties	OK	OK	OK	OK
Instances related through datatype properties whose range is String	OK	OK	OK	OK
Instances related through datatype properties whose range is a XML Schema datatype	OK	-	-	OK

Table 5.14: RDF(S) interoperability results from all the tools to WebODE.

It models and executes. The combinations of components that can be interchanged from the tools that can model the combination of components to WebODE are

- Classes.
- Datatype properties with a domain and whose range is a XML Schema datatype.
- Object properties with a domain and a range.
- Instances of a single class.

- Instances related through object properties, or through datatype properties whose range is *String* or a XML Schema datatype.

It models and fails. The rest of the combinations of components are not interchanged even though WebODE can model them. The reasons for this are the following:

- Class hierarchies without cycles. WebODE does not import the subclass properties if the superclass is not defined as a class in the file.
- Datatype properties with domain and whose range is *String*. WebODE crashes when the *String* range is defined in the file as a datatype without namespace.

It does not model. Some combinations of components of the tool common knowledge model cannot be modelled in WebODE and, therefore, cannot be interchanged with it. Nevertheless, the interoperability experiments from the tools that can model these combinations of components to WebODE provide some information for these combinations of components:

- Classes that are instances of metaclasses. WebODE cannot model metaclasses. Therefore, when it receives metaclasses from other tools, it imports the metaclasses as classes and loses the *rdf:type* properties between classes. If a metaclass is not defined as a class in the exported file, the metaclass is not imported.

If the class is not defined as a class in the exported file, the class is imported as an instance.

- Class hierarchies with cycles. When WebODE finds a cycle in a class hierarchy from Corese, it creates a class and an imported term with the same name as the object of the *rdfs:subClassOf* property that produces the cycle and creates the subclass with the imported term.
- Classes related through object properties. WebODE does not import the property.
- Classes related through datatype properties. WebODE does not import the property.
- Object and datatype properties without domain and without range. When WebODE imports an object or a datatype property without domain and range, it creates *rdfs:Resource* as an imported term and creates the property as an object property with a domain and a range of *rdfs:Resource*.
- Object and datatype properties with domain and without range. When WebODE imports an object or a datatype property with domain but without range, it creates *rdfs:Resource* as an imported term and creates the property as an object property with a range of *rdfs:Resource*.

- Object and datatype properties without domain and with range. When WebODE imports an object or a datatype property without domain but with range, it creates *rdfs:Resource* as an imported term and creates the property with a domain of *rdfs:Resource*.
- Object and datatype properties with undefined resources as domain or range. WebODE creates the undefined resource as a class.
- Object and datatype properties with multiple domains. WebODE imports an object or datatype property that has multiple domains, creating an anonymous concept as the domain of the datatype property and as a subclass of one of the domain classes.
- Object properties with multiple ranges. WebODE imports an object property that has multiple ranges by creating an anonymous concept as the range of the property and as a subclass of one of the range classes.
- Instances of undefined resources. WebODE does not import the instance.
- Instances of multiple classes. WebODE imports the instance as instance of just one class.
- Instances related through undefined object and datatype properties. WebODE does not import the undefined properties.

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

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

5.5.4. Global RDF(S) interoperability results

Since the tools participating in the benchmarking have different knowledge models, both the experiments and the analysis of the results are based on a common group of ontology components that is present in these tools, as shown in section 5.1.3. Therefore, the knowledge models of the tools participating in the benchmarking cover more or less this common group.

Figures 5.14, 5.15 and 5.16 show a quantitative analysis of the global results of executing the RDF(S) Interoperability Benchmark Suite. Each figure corresponds to one tool and shows the results obtained when that tool is the destination of the interchange. For each tool, each figure shows the number of benchmarks that fall into each of the possible combinations of the *Modelling* and *Execution* results defined in section 5.1.3.

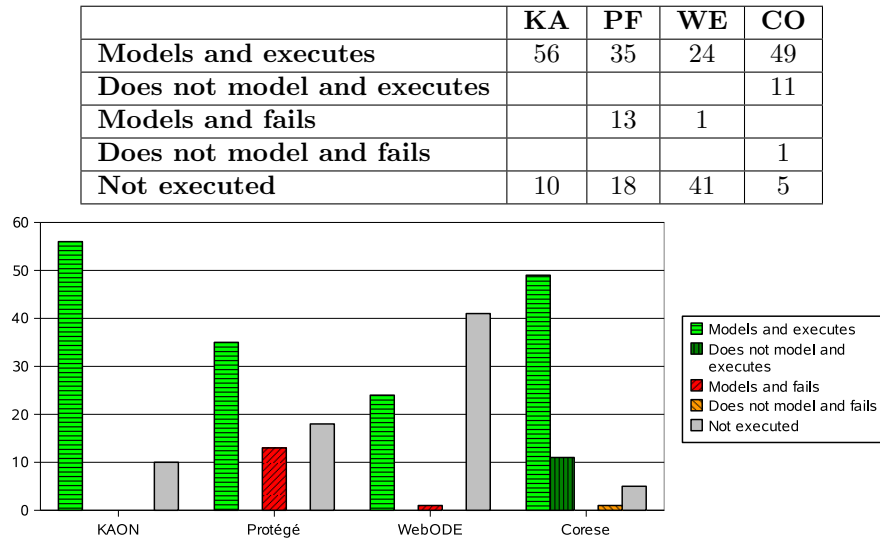


Figure 5.14: RDF(S) interoperability results from all the tools to KAON.

The import and export results presented in the previous sections posed few problems when importing and exporting ontologies; but in the figures mentioned above we can see that all the ontology development tools have interoperability problems both in the components that they are able to model and in those that they are not.

We can say that interoperability between the tools depends on

- The correct working of their RDF(S) importers and exporters.
- The way chosen for serializing the exported ontologies in the RDF/XML syntax.

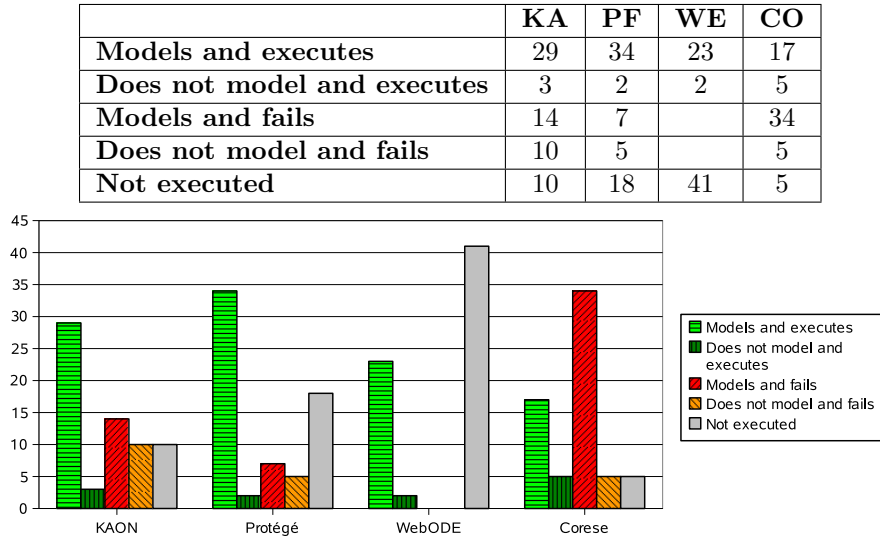


Figure 5.15: RDF(S) interoperability results from all the tools to Protégé-Frames.



Figure 5.16: RDF(S) interoperability results from all the tools to WebODE.

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

If we classify the mismatches found when comparing the interchanged ontology with the expected one according to the levels of ontology translation problems described in section 1.3.3, we can see that most of the mismatches belong to the *Conceptual* level (i.e., they are due to differences in conceptualizations). The only exceptions are the cases in which the benchmarks that deal with component naming restrictions produce mismatches at the *Lexical* level; and this is so because in some cases tools do not encode component names in the translation.

Next, the components that can be interchanged between the tools are listed in table 5.15. The table illustrates the different combinations of components classified into categories, and each column shows whether the combination of components in that category can be interchanged between a group of tools¹¹.

In the table abovementioned, “Y” means that all the benchmarks in the category have an *Execution* value of *OK*, “N” means that at least one of the benchmarks in the category has an *Execution* value of *FAIL*, and the “-” character means that the component cannot be modelled in some of the tools and, therefore, cannot be interchanged between them.

It must be noted that a benchmark can be part of several categories. For example, benchmark *In35* (Interchange just one object property that has as domain several classes, with the classes defined in the ontology) belongs to the “Object properties without domain or range” and to the “Object properties with multiple domains or ranges” categories.

Interoperability using the same tool

Tools pose no problems when they interchange with themselves ontologies that contain components of the subset of the knowledge models considered in the benchmarks (i.e., a tool exports an ontology to an RDF(S) file and this file is imported by the same tool). The only exception is Protégé-Frames, as we will see below.

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

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

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

¹¹The names of the tools have been shortened in the heading of the table: K=KAON, P=Protégé-Frames and W=WebODE

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

Table 5.15: Combinations of components interchanged between the tools.

Interoperability between each pair of tools

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

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

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

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

Interoperability between all the tools

Interoperability between **KAON**, **Protégé-Frames** and **WebODE** can be achieved through nearly all the common components that all these tools can model: classes, class hierarchies without cycles, object properties with a domain and a range, instances of a single class, and instances related through object properties. The only common components that these tools cannot use are datatype properties with domain and whose range is *String* and instances related through datatype properties.

Interoperability regarding URI character restrictions

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

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

Protégé-Frames cannot interchange with itself, nor with **KAON**, nor with **WebODE** ontologies that have URI character restrictions in class and property names.

WebODE can interchange with itself ontologies that have class and property names but that do not start with a letter, nor with “_”, and nor with spaces in their names.

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

OWL Interoperability Benchmarking

This chapter presents the instantiation of the benchmarking methodology for the tasks of the *Experiment* phase of the OWL Interoperability Benchmarking. It contains a complete definition of the automatic approach of the UPM-FBI and of the benchmark suite used in the experiment: the OWL Lite Import Benchmark Suite (figure 6.1); it also provides information about how the experiments were executed over the tools that participated in the benchmarking, namely, GATE (version 4.0), Jena (version 2.3), KAON2 (version 2006-09-22), Protégé-Frames (version 3.3 build 395), Protégé-OWL (version 3.3 build 395), SemTalk (version 2.3), SWI-Prolog (version 5.6.35), and WebODE (version 2.0 build 140) and a detailed analysis of the results for these tools; finally, it shows an example of the improvement of the OWL interoperability results after debugging one of the tools participating in the benchmarking.

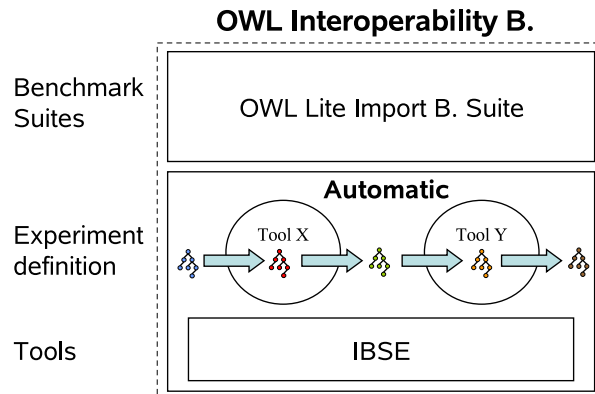


Figure 6.1: UPM-FBI resources for benchmarking OWL interoperability.

The chapter is structured as follows: sections 6.1 and 6.2 present the defi-

inition of the experiment and of the benchmark suites used in it, respectively. Section 6.3 describes the IBSE tool, i.e., the evaluation infrastructure that automates the execution of the experiments. Subsequently, section 6.4 presents some results about the OWL compliance results of the tools and section 6.5 describes the results of performing the experiment over the different tools participating in the benchmarking. Finally, section 6.6 shows how the OWL interoperability results of WebODE were improved after debugging the tool.

6.1. Experiment definition

The experiments to be performed must provide data informing how the Semantic Web tools comply with the evaluation criteria defined in section 4.4:

- The components of a tool knowledge model 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 arise when ontologies are interchanged between two tools and the causes of these problems.

As mentioned before, participation in the benchmarking is open to any Semantic Web tool. Nevertheless, the experiment requires that the tools participating be able to import and export OWL ontologies. This is so because in the experiment 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. Due to 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 an ontology from a file, and to export an ontology to a file.

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

Each benchmark execution comprises two sequential steps, shown in Figure 6.2. Let start with a file containing an OWL ontology (O_i). The first step (*Step 1*) consists in importing the file with the ontology into the origin tool and then exporting the ontology to an OWL file (O_i^I). The second step (*Step 2*) consists in importing the file with the ontology exported by the origin tool (O_i^I) into the destination tool and then exporting the ontology to another file (O_i^{IV}).

In these steps there is not a common way of checking how good the importers (by comparing O_i with O_i^I and O_i^{II} with O_i^{III}) and exporters (by comparing

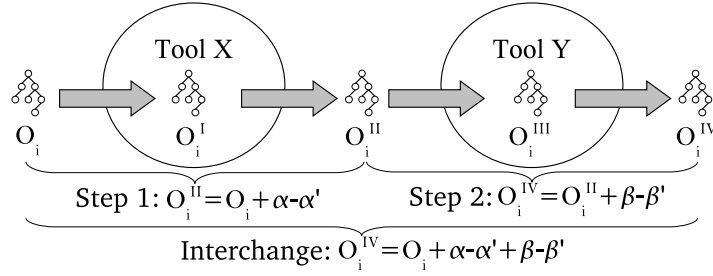


Figure 6.2: The two steps of a benchmark execution.

O_i^I with O_i^{II} and O_i^{III} with O_i^{IV}) are. We just have the results of combining the import and export operations (the files exported by the tools), so these two operations are viewed as an atomic operation. It must be noted, therefore, that if a problem arises in one of these steps, we cannot know whether it was originated when the ontology was being imported or exported because we do not know the state of the ontology inside each tool.

After a benchmark execution, we have three ontologies to be compared, namely, the original ontology (O_i), the intermediate ontology exported by the first tool (O_i^I), and the final ontology exported by the second tool (O_i^{IV}). From these results, the following three evaluation criteria for a benchmark execution can be defined:

- **Execution** (*OK/FAIL/C.E./N.E.*) informs of the correct execution of a step or of 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 it compares the original and the final ontologies; and *N.E.* (Not Executed) if the second step is not executed because the first step execution failed.
- **Information added or lost** shows 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*) explains 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.*.

For evaluating the interoperability of the tools, the OWL Lite Import Benchmark Suite has been used (described in the following section), which is common to all the tools and contains ontologies with simple combinations of OWL Lite components.

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 experiment. Hence, we need some software application that can perform all the experiments automatically.

Such software application is IBSE¹ (Interoperability Benchmark Suite Executor), which 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 of the specific procedure to follow for using it are detailed in section 6.3.

6.2. The OWL Lite Import Benchmark Suite

The ontologies used in the experiment are those defined for the OWL Lite Import Benchmark Suite, which is described in detail in [David *et al.*, 2006]. This benchmark suite was intended to evaluate the OWL import capabilities of Semantic Web tools by checking the import of ontologies with simple combinations of components of the OWL Lite knowledge model. This benchmark suite is composed of 82 benchmarks and is available in the Web².

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

- To have a small number of benchmarks. Benchmarking is a process that consumes a lot of resources, and any increase in the number of benchmarks leads to an increment in the time required for performing the experiments and for subsequently analysing the results.
- To use OWL Lite to define the ontologies in order to limit the number of benchmarks. Furthermore, annotation, versioning and heading vocabulary terms are not considered.
- To use the RDF/XML syntax³ 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

¹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

²http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html

³<http://www.w3.org/TR/rdf-syntax-grammar/>

the syntactic correctness of the ontologies, we decided to use an OWL validator⁴.

- To define simple ontologies only. This will allow detecting problems easily in the tools.

Since the RDF/XML syntax allows serializing ontology components in different ways while maintaining the same semantics, the benchmark suite includes two types of benchmarks: one that checks the import of the different combinations of the OWL Lite vocabulary terms, and another that checks the import of OWL ontologies with the different variants of the RDF/XML syntax.

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

6.2.1. Benchmarks that depend on the knowledge model

The process 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, and finally we wrote them in the RDF/XML syntax.

When defining the ontologies, we contemplated the different possibilities of defining in OWL Lite 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).

We also decided to discard those vocabulary terms that do not contribute to the OWL expressiveness, i.e., annotation, versioning, and heading vocabulary terms.

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 ones. When the number of the ontologies defined was large, we pruned the benchmark suite. We also decided to take into account 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.

Appendix C presents how the ontologies were defined for the benchmarks in each group; it also provides the vocabulary terms and the productions (axioms) involved.

6.2.2. 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.

⁴<http://phoebus.cs.man.ac.uk:9999/OWL/Validator>

The 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.

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; hence, we have defined a benchmark for each possibility:

- 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. Here benchmarks are defined 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 though the second one is more compactly written.

```

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

```

6.2.3. Description of the benchmarks

Each benchmark of the benchmark suite, as table 6.1 shows, is described with a unique **identifier**, 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 syntax⁵.

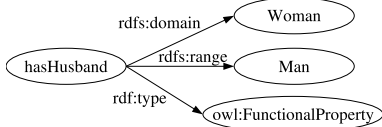
Identifier	ISG03
Description	Import a single functional object property whose domain is a class and whose range is another class
Formal description	$\top \sqsubseteq \leq 1 \text{ hasHusband}$ $\top \sqsubseteq \forall \text{hasHusband}^-. \text{Woman}$ $\top \sqsubseteq \forall \text{hasHusband}. \text{Man}$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

Table 6.1: An example of an OWL import benchmark definition.

⁵All the files have been syntactically validated against the WonderWeb OWL Ontology Validator (<http://phoebus.cs.man.ac.uk:9999/OWL/Validator>)

The OWL Lite Import Benchmark Suite is available in a public web page⁶ and is composed of 82 benchmarks that are classified into 12 groups, each identified by one letter (from **A** to **L**). Table 6.2 shows the 12 groups and the number of benchmarks that each group contains.

Group	No.
A - Class hierarchies	17
B - Class equivalences	12
C - Classes defined with set operators	2
D - Property hierarchies	4
E - Properties with domain and range	10
F - Relations between properties	3
G - Global cardinality constraints and logical property characteristics	5
H - Single individuals	3
I - Named individuals and properties	5
J - Anonymous individuals and properties	3
K - Individual identity	3
L - Syntax and abbreviation	15
TOTAL	82

Table 6.2: Groups of the OWL import benchmarks.

The list of all the benchmarks composing the benchmark suite can be found in appendix D; 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 D.2.

The OWL Lite Import Benchmark Suite has been used to evaluate the interoperability of Semantic Web tools. However, any group of ontologies could have been used as input for the experiment. For example, we could have employed a group of real ontologies in a certain domain, ontologies synthetically generated such as the Lehigh University Benchmark (LUBM) [Guo *et al.*, 2005] or the University Ontology Benchmark (UOB) [Ma *et al.*, 2006], or the OWL Test Cases⁷ (developed by the W3C Web Ontology Working Group).

These ontologies were designed with specific goals and requirements, such as that of performance evaluation or correctness evaluation. However, since our goal was to improve interoperability, these ontologies could complement our experiments but, in our circumstances, we aimed at evaluating interoperability with simple OWL ontologies that, even though they do not cover exhaustively the OWL specification, are simple and allow isolating problem causes and highlighting problems in the tools.

⁶http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html

⁷<http://www.w3.org/TR/owl-test/>

6.2.4. Towards benchmark suites for OWL DL and Full

Although the OWL Lite Import Benchmark Suite described in this section 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 which are not allowed in OWL Lite. In addition, the use of the OWL vocabulary terms is restricted both in OWL Lite and in OWL DL. Hence, the benchmark suite defined for OWL Lite is incomplete for OWL DL and OWL Full.

The sections below analyse the possibility of extending the OWL Lite Import Benchmark Suite with the object of covering OWL DL and OWL Full, examining the differences between the three species of OWL. However, the definition of these extensions is out of the scope of this work.

OWL DL

As 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 6.3 shows, to cover the OWL DL sublanguage of OWL, we should also need to take into account

- The different combinations of the OWL Lite vocabulary terms according to their use in OWL DL, since OWL DL imposes fewer restrictions. Table 6.3 shows the differences in the restrictions of use of the vocabulary terms for OWL Lite and DL⁸.
- The different combinations of the OWL DL vocabulary terms not allowed in OWL Lite, with themselves and with the OWL Lite vocabulary terms. The vocabulary terms allowed in OWL DL that are not allowed in OWL Lite are the following: *owl:hasValue*, *owl:disjointWith*, *owl:unionOf*, *owl:complementOf*, *owl:DataRange*, and *owl:oneOf*.

For example, if we wanted to extend the benchmarks for *owl:equivalentClass* and *rdfs:subClassOf*, we should define new benchmarks that have 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

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

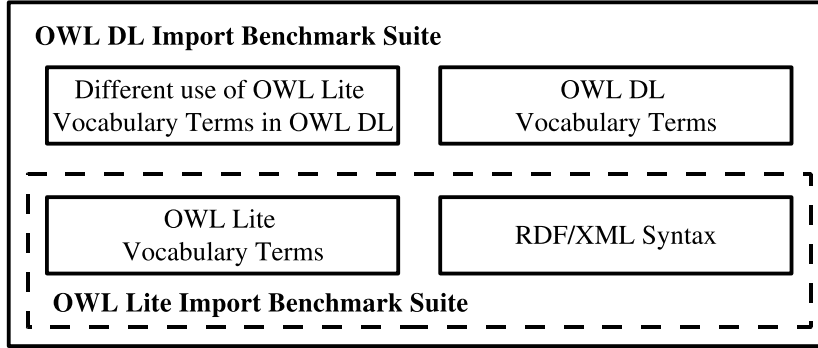


Figure 6.3: The OWL DL Import Benchmark Suite.

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 ≥ 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 6.3: Restrictions in the use of OWL Lite and OWL DL.

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 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.

Following this approach, a considerable part of the benchmarks could be reused without any modification and, therefore, any tool that had already per-

formed the experiments of the OWL Lite Import Benchmark Suite would not need to repeat them.

Nevertheless, the restrictions of use of the OWL vocabulary terms in OWL DL are more relaxed than in OWL Lite; therefore, a larger number of new benchmarks would be defined and this which would affect the usability of the whole benchmark suite.

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 the following:

- All the RDF(S) vocabulary can be used within OWL Full.
- OWL Full has no separation among 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 are 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 take into account 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 (section 5.1).

6.3. Experiment execution: the IBSE tool

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 executing the experiments between any selected group of tools and of analysing the results and permits smoothly including new tools into the infrastructure.

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

6.3.1. IBSE requirements

The main requirements for 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 both the experiment execution and the analysis of the results.** In the OWL Interoperability Benchmarking we sacrifice a higher detail in results in order to prevent the experiments from being conducted by humans. 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 in order to know the cause of the problems encountered and to extract 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 the time.
- **To use any group of ontologies as input for the experiments.** Executing benchmarks with no human effort can provide further advantages. Therefore, 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.

6.3.2. IBSE implementation

A normal execution of IBSE comprises the three consecutive steps shown in figure 6.4, even though they can also be executed independently. These steps are the following:

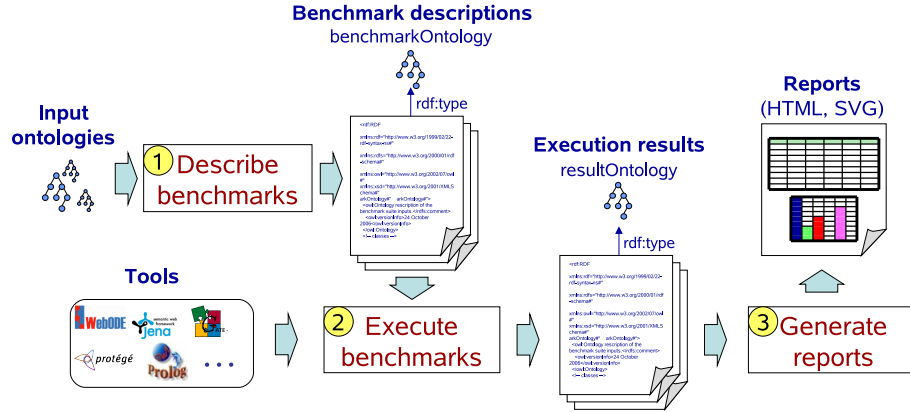


Figure 6.4: Automatic experiment process in IBSE.

1. **To generate machine-readable benchmark descriptions from a group of ontologies.** In this step, and from a group of ontologies located in a URI, one RDF file with one benchmark for each ontology is generated, using 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, taking into account all the different combinations of ontology interchanges between the tools, each benchmark described in the RDF file is executed and its results are stored in another RDF file, employing the vocabulary of the *resultOntology* ontology.

To execute a benchmark between an origin tool and a destination one, as described in section 6.1, first the file with the ontology is imported into the origin tool and then exported into an intermediate file and, second, this intermediate file is imported into the destination tool and then exported into the final file.

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 6.1. This comparison and its output depend on an external ontology comparer. The current implementation

uses the OWL comparer of the *KAON2 OWL Tools*⁹, although other comparers can be inserted by implementing a Java interface.

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.

Representation of benchmarks and results

This section describes the two OWL ontologies employed in the IBSE tool: the *benchmarkOntology*¹⁰ ontology, which defines the vocabulary that represents the benchmarks to be executed, and the *resultOntology*¹¹ 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 E using the RDF/XML syntax.

Figures 6.5 and 6.6 show the graphical representation of the *benchmarkOntology* and of the *resultOntology* ontologies, respectively. 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*.

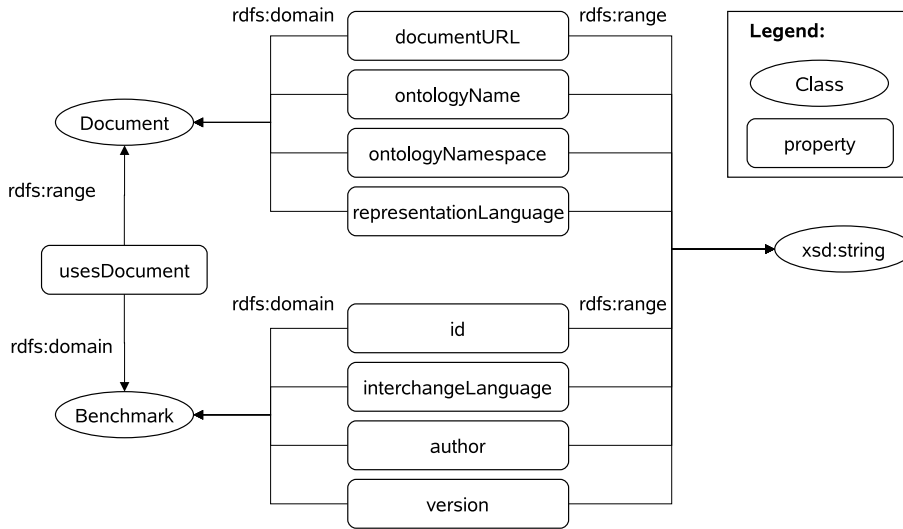
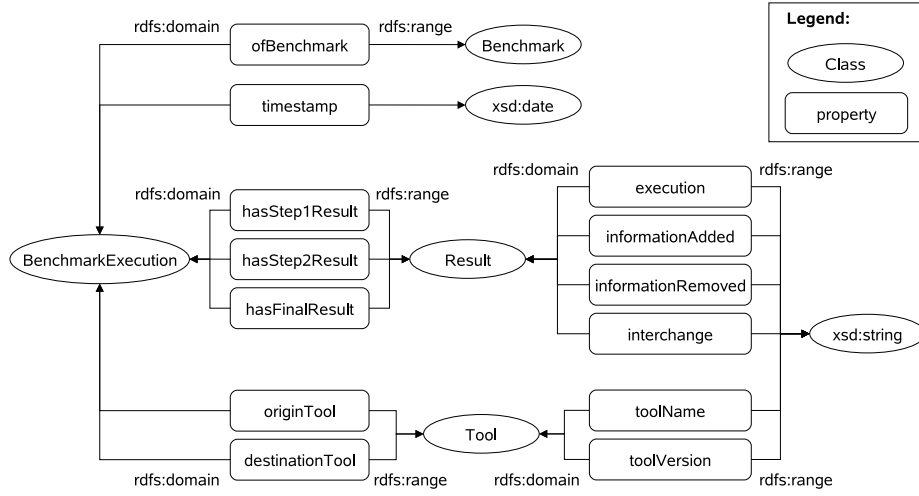


Figure 6.5: Graphical representation of the *benchmarkOntology* ontology.

⁹version 0.27 <http://owltools.ontoware.org/>

¹⁰http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/benchmarkOntology.owl

¹¹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/resultOntology.owl

Figure 6.6: Graphical representation of the *resultOntology* ontology.

benchmarkOntology. The *Document* class represents a document containing one ontology. A document can be further described by properties that have *Document* as domain. Such properties are the following: *documentURL* (the URL of the document), *ontologyName* (the ontology name), *ontologyNamespace* (the ontology namespace), and *representationLanguage* (the language used to implement the ontology).

The *Benchmark* class represents a benchmark to be executed. A benchmark can be further described with properties that have *Benchmark* as domain. Such properties are the following: *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 properties that have *Tool* as domain. Such properties are the following: *toolName* (the tool name), and *toolVersion* (the tool version number).

The *Result* class represents a result of one step or of the whole benchmark execution. A result can be further described with properties that have *Result* as domain. Such properties are the following: *execution* (if the whole interchange, the first step or the second step are carried out without any execution problem); *informationAdded* (the triples added in the whole interchange, in the first step, or in the second step); *informationRemoved* (the triples removed in the whole interchange, in the first step, or in the second step); and *interchange* (if the ontology has been

interchanged correctly from the origin tool to the destination tool, in the first step or in the second step with no addition or loss of information).

The *BenchmarkExecution* class represents a result of a benchmark execution. A benchmark execution can be further described with properties that have *BenchmarkExecution* as domain. Such properties are the following: *ofBenchmark* (the benchmark to which the result corresponds); *originTool* (the tool origin of the interchange); *destinationTool* (the tool destination of the interchange); and finally, *timestamp* (the date and time when the benchmark is executed).

Inserting a new tool

As the experiment requires no human intervention, we can only insert new tools by accessing them through application programming interfaces (APIs) or through batch executions. There are other ways of executing an application automatically (e.g., Web Service executions) but they are not present in the current tools. Nevertheless, to adapt the IBSE tool in order to include other types of executions should be quite straightforward.

Inserting a new tool in the evaluation infrastructure is quite easy, it can be performed either by implementing a Java interface in IBSE or by building a program that imports an ontology from a file and exports 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, i.e., GATE, Jena, KAON2, the NeOn Toolkit, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE.

Most of these tools have implemented the Java interface, because they provide Java methods for performing the import and export operations. In the case of non-Java tools (SemTalk and SWI-Prolog), these operations were performed by executing precompiled binaries.

As an example, figure 6.7 shows the implementation of the *ImportExport* method for Jena.

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 an RDF(S) comparer (rdf-utils¹² version 0.3b) and of an OWL

¹²<http://wymiwyg.org/rdf-utils/>

```

public void ImportExport(String importFile, String exportFile, String ontologyName,
                        String namespace, String language)
    throws BadURIException
{
    // Create the Jena models
    Model model = ModelFactory.createDefaultModel();
    Model model_out = ModelFactory.createDefaultModel();

    try {
        // Import the ontology in the file into a Jena model
        FileInputStream inFile = new FileInputStream(importFile);
        model = model.read(importFile, null, null);
        inFile.close();

        // Export the contents of the model into a file
        FileOutputStream outFile = new FileOutputStream(exportFile);
        String queryString = "DESCRIBE ?x WHERE {?x ?y ?z}";
        Query query = QueryFactory.create(queryString);
        QueryExecution qexec = QueryExecutionFactory.create(query, model);
        model_out = qexec.execDescribe();
        model_out.write(outFile);

        // Close the models
        model.close();
        model_out.close();

    } catch (FileNotFoundException e) { e.printStackTrace(); }
    catch (IOException e) { e.printStackTrace(); }
}

```

Figure 6.7: Implementation of the *ImportExport* method for Jena.

comparer (KAON2 OWL Tools¹³ 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.

Since the software used for ontology comparison could have execution problems, we needed a previous evaluation of this software to ensure the validity of the benchmarking results.

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

1. The interoperability experiment was carried out with the tools that have OWL as knowledge model, because these tools should interchange all the ontologies correctly since 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. In this

¹³<http://owltools.ontoware.org/>

step, we analysed the cases where the comparison of two ontologies caused an execution error in the comparer.

After carrying out the previous steps we found several problems in the KAON2 OWL Tools ontology comparer. These problems were the following:

- When one of the ontologies was empty, the comparer returned that the ontologies were the same.
- The comparer returned 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 returned 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 compared two ontologies with blank nodes, it generated different node identifiers and, therefore, it returned that the ontologies were different.
- When one of the ontologies was not a valid OWL ontology in the RDF/XML syntax, the comparer threw an exception.
- The comparer is not robust and threw an exception when it compared ontologies with unexpected inputs, as for example, the incorrect class naming produced by some tools¹⁴, or the incorrect use of the OWL language constructors, i.e., 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 *rdfs:subclassOf* statements with no object ("*<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 other cases was documented and 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.

¹⁴i.e., "*#http-3A_2F_2Fwww.w3.org_2F2002_2F07_2Fowl_23Thing*"

6.3.3. Using IBSE

The IBSE tool has been implemented in Java; its source code and binaries are publicly available and can be downloaded from its web page¹⁵. The only requirements for executing IBSE are to have a Java Runtime Environment and the IBSE binaries. To perform the experiments with SemTalk and WebODE, they must also be installed in the system. The latest version of the IBSE source code is located in a Subversion repository¹⁶.

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 or binaries; others, however, do need preparation (e.g., WebODE must be running in order for IBSE to access it).
4. To run IBSE from the command line: *java -jar IBSE.jar [config file]*.

Steps 2. and 3. are optional for the default execution of the experiments and for the generation of the reports. Nevertheless, the *ibse.conf* file allows customizing the execution by defining¹⁷

- The tools considered as the origin and the destination of the interchange (*ORIGIN_TOOLS* and *DESTINATION_TOOLS*).
- The language used in the input ontologies and in the interchange (*REPRESENTATION_LANGUAGE*).
- The steps to perform in the execution (*DESCRIBE_BENCHMARKS*, *EXECUTE_BENCHMARKS*, *GENERATE_REPORT_FROM*).
- The location of the data that is needed or generated by IBSE (*ONTOLOGIES_URL*, *BENCHMARKS_URL*, *RESULTS_URL*, *RESULTS_RDF_URL*, *RESULTS_HTML_URL*).

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 Web¹⁸.

¹⁵http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

¹⁶http://delicias.dia.fi.upm.es/repos/interoperability_benchmarking/

¹⁷Full use details of use can be found in the comments of the *ibse.conf* file.

¹⁸[http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/OIBS.](http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/OIBS.rdf)

- RDF files (*Result<Tool1><Tool2>.rdf*) with the descriptions of the results for each pair of tools.
- The ontologies resulting from executing the experiments, including the intermediate and final ones.
- HTML files with different visualizations, summaries and statistics of the results, such as the following:
 - 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.

6.4. OWL compliance results

Once the IBSE tool was adapted to include all the tools participating in the benchmarking, the experiments were performed using the ontologies from the OWL Lite Import Benchmark Suite. As mentioned in section 4.4.1, results were obtained for eight tools: GATE, Jena, KAON2, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE.

The author conducted the experiments, compiled all the execution results, made these results (the HTML and RDF files) available in the benchmarking web page¹⁹, and provided a detailed analysis of them, including results specific to each tool; this information can be found at [García-Castro *et al.*, 2007a].

As the OWL interoperability results highly depend on the compliance of the tools with the OWL specification and because of the large number of benchmark executions²⁰, the analysis of the interoperability of the tools is divided into two consecutive steps:

¹⁹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/2007-08-12_Results/

²⁰For 9 tools we have 81 possible interoperability scenarios, each composed of 82 benchmark executions, which results in 6.642 benchmark executions.

1. The analysis of the compliance of the tools with the OWL specification, taking into account the results of the tool when managing OWL ontologies in the combined operation of importing an OWL ontology and exporting it again (a step of the experiment, as defined in section 6.1). This analysis is included in this section.
2. The analysis of the OWL interoperability of the tools with all the tools participating in the benchmarking. This analysis is presented in section 6.5.

To analyse the OWL compliance of the tools, we have contemplated the tool results when such tool is the origin of the interchange (*Step 1*), irrespective of the tool being the destination of the interchange. This step has as input an original ontology that is imported by the tool (O_i) and then exported into a resultant ontology (O_i^{II}). This analysis has been performed by comparing the original and the resultant ontologies.

This is not an exhaustive evaluation of the OWL compliance because the ontologies used in the experiments belong to the OWL Lite sublanguage and do not represent any possible combination of ontology components. However, it provides useful information about the behaviour of the tools when dealing with OWL ontologies.

First, a summary of the OWL compliance of each tool is presented. Second, an analysis of the OWL compliance of the tools from a global viewpoint is presented.

In these analyses, we have disregarded the 15 benchmarks of the *Syntax and abbreviation* group because the working of the importers when dealing with the different syntax variants cannot be properly measured as the results also include the effect of the exporters. Therefore, we have focused on the combinations of components present in the ontology and the results provide data of the 67 benchmarks of groups A to K instead of the 82 benchmarks of the benchmark suite.

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

6.4.1. GATE OWL compliance results

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 (even though the ontology validates correctly).

The results of a step execution in GATE, as shown in figure 6.8, 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*).

- *Execution fails when the ontologies are compared.* This occurs in 1 case (ISH02).

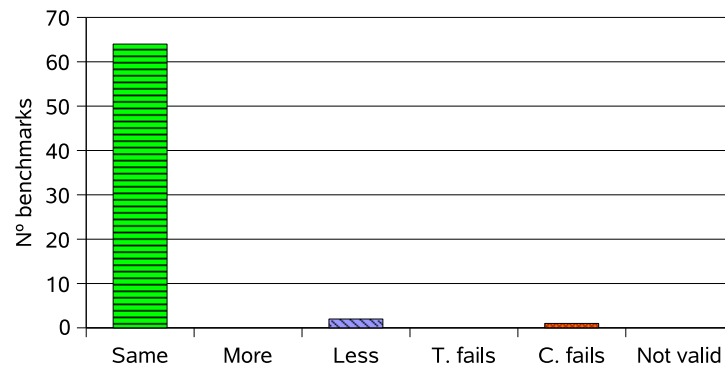


Figure 6.8: OWL import and export operation results for GATE.

Below, the behaviour of GATE in one step is described, focusing on the combination of components present in the original ontology.

Class hierarchies

- Named class hierarchies without cycles. 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. The ontologies processed remain the same.
- Classes that are a subclass of a value constraint in an object property. 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. The ontologies processed remain the same.
- Classes that are a subclass of a class intersection. The ontologies processed remain the same.

Class equivalences

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

- Classes equivalent to a cardinality constraint in an object or datatype property. The ontologies processed remain the same.
- Classes equivalent to a class intersection. The ontologies processed remain the same.

Classes defined with set operators

- Classes that are intersection of other classes. The ontologies processed remain the same.

Properties

- Object and datatype property hierarchies. The ontologies processed remain the same.
- Object and datatype properties with or without domain or range, or with multiple domains or ranges. The ontologies processed remain the same.

Relations between properties

- Equivalent object and datatype properties. The ontologies processed remain the same.
- Inverse object properties. The ontologies processed remain the same.

Global cardinality constraints and logical property characteristics

- Transitive, symmetric, or inverse functional object properties. The ontologies processed remain the same.
- Functional object and datatype properties. The ontologies processed remain the same.

Individuals

- Individuals of a single class. One of the instances is lost.
- Individuals of multiple classes. The comparer launches an exception but the ontologies processed remain the same.
- Named individuals and object or datatype properties. The ontologies processed remain the same.
- Anonymous individuals and object or datatype properties. 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, which implies that the ontologies are different.

Individual identity

- Equivalent or different individuals. The ontologies processed remain the same.

6.4.2. Jena OWL compliance results

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 6.9.

When there are anonymous individuals and object or datatype properties, 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, which implies that the ontologies are different.

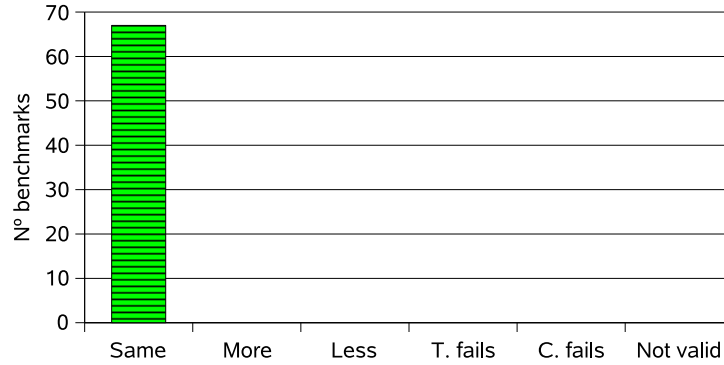


Figure 6.9: OWL import and export operation results for Jena.

6.4.3. KAON2 OWL compliance results

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 (even though the ontology validates correctly).

The results of a step execution in KAON2, as shown in figure 6.10, 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).

- *Execution fails when the ontologies are compared.* This occurs in 8 cases (ISA09, ISA12-13, ISA16, ISB04, ISB07-08, ISB11).

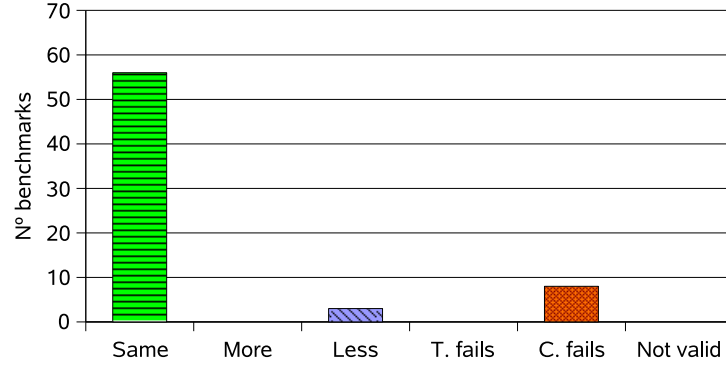


Figure 6.10: OWL import and export operation results for KAON2.

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

Class hierarchies

- A single class. The class is lost.
- Named class hierarchies with or without cycles. The ontologies processed remain the same.
- Classes that are a subclass of a value constraint in an object property. The ontologies processed remain the same.
- Classes that are a subclass of an *owl:maxCardinality* or *owl:cardinality* cardinality constraint in an object or datatype property. The ontologies processed remain the same.
- Classes that are a subclass of an *owl:minCardinality* cardinality constraint in an object or datatype property. 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. The ontologies processed remain the same.

Class equivalences

- Classes equivalent to named classes. The ontologies processed remain the same.

- Classes equivalent to a value constraint in an object property. The ontologies processed remain the same.
- Classes equivalent to an *owl:maxCardinality* or *owl:cardinality* cardinality constraint in an object or datatype property. The ontologies processed remain the same.
- Classes equivalent to an *owl:minCardinality* cardinality constraint in an object or datatype property. The class is created as equivalent to a blank node instead of being created as equivalent to the restriction. The class is considered an instance (*Individual(a:Employee value(owl:equivalentClass ""))*). *owl:equivalentClass* is used as a datatype property (OWL Full).
- Classes equivalent to a class intersection. The ontologies processed remain the same.

Classes defined with set operators

- Classes that are intersection of other classes. The ontologies processed remain the same.

Properties

- Object and datatype property hierarchies. 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. The ontologies processed remain the same.

Relations between properties

- Equivalent object and datatype properties. The ontologies processed remain the same.
- Inverse object properties. The ontologies processed remain the same.

Global cardinality constraints and logical property characteristics

- Transitive, symmetric, or inverse functional object properties. The ontologies processed remain the same.
- Functional object and datatype properties. The ontologies processed remain the same.

Individuals

- Individuals of a single or multiple classes. The ontologies processed remain the same.

- Named individuals and object or datatype properties. The ontologies processed remain the same.
- Anonymous individuals and object or datatype properties. The ontologies processed remain the same.

Individual identity

- Equivalent or different individuals. The ontologies processed remain the same.

6.4.4. Protégé-Frames OWL compliance results

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

The results of a step execution in Protégé-Frames, as shown in figure 6.11, 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).
- *Execution fails when the ontologies are compared.* This occurs in 12 cases (ISA13-16, ISB08-11, ISE07, ISI04-05, ISJ03).

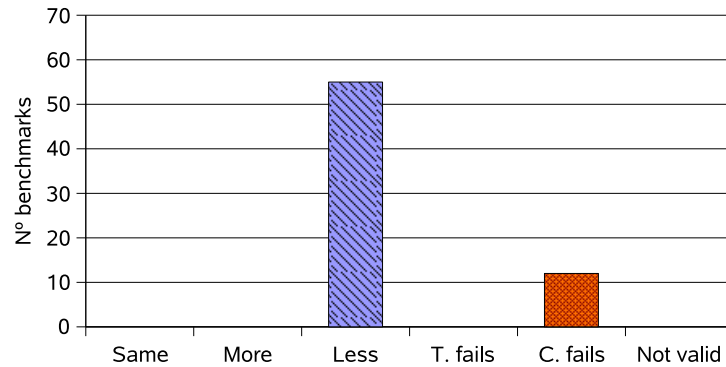


Figure 6.11: OWL import and export operation results for Protégé-Frames.

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

Class hierarchies

- Classes. Whenever classes appear, the names of the classes are changed from “<class_name>” to “ibs_<class_name>” and an *rdfs:label* is inserted into the classes with the value “*ibs:<class_name>”[^]xsd:string*”.
- Named class hierarchies without cycles. Classes are defined as a subclass of *owl:Thing*.
- Named class hierarchies with cycles. When there are multiple classes, the classes are defined as equivalent.
- Classes that are a subclass of a value constraint in an object property. 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. 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. The datatype properties are changed into *rdf:Property* and 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 a class is constrained by *owl:maxCardinality* and *owl:minCardinality*, the class is 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. The *owl:intersectionOf* property is lost but the ontologies are equivalent.

Class equivalences

- Classes equivalent to named classes. Classes are defined as a subclass of *owl:Thing*.
- Classes equivalent to a value constraint in an object property. 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. Properties are created with a domain. Classes are defined as a subclass of *owl:Thing*. Classes are also 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. The datatype properties are changed into *rdf:Property* and 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. 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 that are intersection of other classes. The *owl:intersectionOf* property is lost. The classes of the intersection are defined as a subclass of the class.

Properties

- Object and datatype properties. The names of the properties are changed from “<property_name>” to “*ibs_<property_name>*”. An *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. The *rdfs:subPropertyOf* property is lost.
- Datatype property hierarchies. The datatype properties are changed into *rdf:Property*. The *rdfs:subPropertyOf* property is lost.
- Object properties with or without domain or range. No further issues have been identified besides those mentioned for object and datatype properties.
- Object properties with multiple domains or ranges. When there are object properties with multiple domains, all domains except one are lost.
- Datatype properties without domain or range. The datatype properties are changed into *rdf:Property*.
- Datatype properties with domain and range. The datatype properties are changed into object properties.
- Datatype properties with multiple domains. The datatype properties are changed into object properties. All domains except one are lost.

Relations between properties

- Equivalent object and datatype properties. The *owl:equivalentProperty* property is lost.
- Inverse object properties. 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. The transitivity and the symmetry are lost.
- Functional object and datatype properties. The datatype properties are changed into object properties.
- Inverse functional object properties. The inverse functionality is lost.

Individuals

- Individuals. The names of individuals are changed from “<individual_name>” to “ibs_<individual_name>”. An *rdfs:label* is inserted into the individuals with the value “*ibs:<name>*”[^]*xsd:string*. This occurs whenever individuals appear.
- Individuals of a single class. The individuals remain the same.
- Individuals of multiple classes. All the type properties except one are lost.
- Named individuals and object or datatype properties. When there are named individuals and datatype properties, datatype properties are changed into object properties.
- Anonymous individuals and object or datatype properties. The anonymous individual is created as a named individual. When there are named individuals and datatype properties, datatype properties are changed into object properties.

Individual identity

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

6.4.5. Protégé-OWL OWL compliance results

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 6.12.

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 they are semantically the same. The only change is that Protégé-OWL defines the *owl:inverseOf* property in both properties instead of in just one.

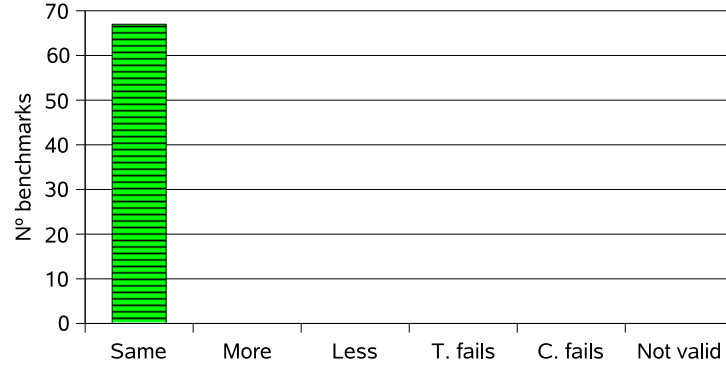


Figure 6.12: OWL import and export operation results for Protégé-OWL.

6.4.6. SemTalk OWL compliance results

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 (even though these ontologies validate correctly).

The results of a step execution in SemTalk, as shown in figure 6.13, 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).
- *Execution fails when the ontologies are compared.* This occurs in 8 cases (ISA09-12, ISB04-07).

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

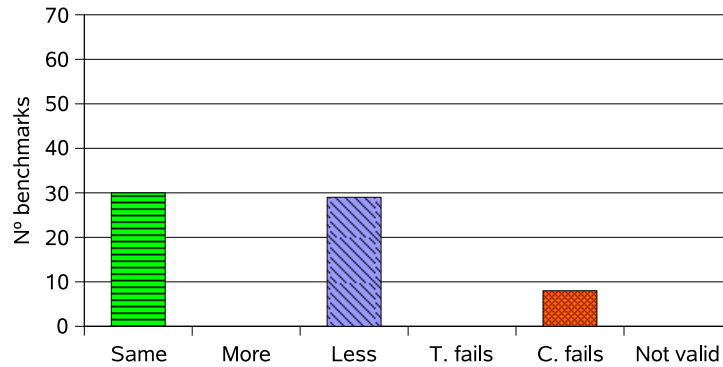


Figure 6.13: OWL import and export operation results for SemTalk.

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. The named class hierarchies remain the same.
- Named class hierarchies with cycles. 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. 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. The object property is defined both as an object and as a datatype property. The class is defined as a subclass of the *restriction(a:hasName value ("^^xsd:string))* restriction. 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. The class is defined as a subclass of the *restriction(a:hasName value*

(`"^^xsd:string`) restriction. 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. The ontologies processed remain the same.

Class equivalences

- Classes equivalent to named classes. The *owl:equivalentClass* property is lost.
- Classes equivalent to a value constraint in an object property. 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. 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, it is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.
- Classes equivalent to a cardinality constraint in a datatype property. Classes are defined as a subclass instead of being defined as equivalent to the restriction. In the case of the *owl:cardinality* constraint, it is replaced by one *owl:minCardinality* constraint and one *owl:maxCardinality* constraint.
- Classes equivalent to a class intersection. The *owl:intersectionOf* property is lost.

Classes defined with set operators

- Classes that are intersection of other classes. The ontologies processed remain the same.

Properties

- Object and datatype property hierarchies. The ontologies processed remain the same.
- Object properties with or without domain or range or with multiple domains and ranges. The ontologies processed remain the same.
- Datatype properties with or without domain or range or with multiple domains. The range is lost.

Relations between properties

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

Global cardinality constraints and logical property characteristics

- Transitive or symmetric object properties. The ontologies processed remain the same.
- Functional object and datatype properties. When there are datatype properties, the range is lost and also lost is the statement about the property being functional.
- Inverse functional object properties. The statement about the property being inverse functional is lost.

Individuals

- Individuals of a single or multiple classes. The ontologies processed remain the same.
- Named individuals and object or datatype properties. When there are datatype properties, the range is lost.
- Anonymous individuals and object or datatype properties. The anonymous individual is lost.

Individual identity

- Equivalent or different individuals. 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.

6.4.7. SWI-Prolog OWL compliance results

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 6.14.

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.

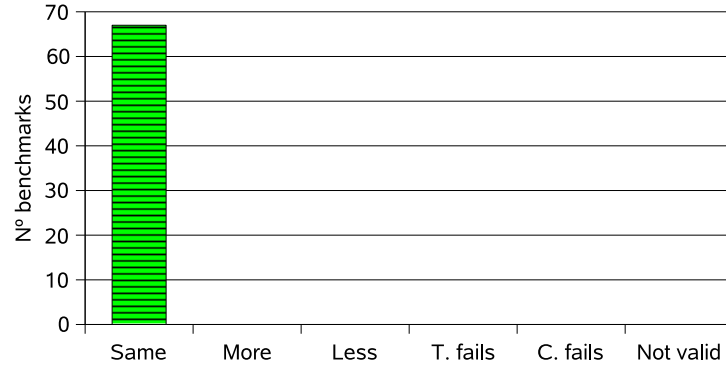


Figure 6.14: OWL import and export operation results for SWI-Prolog.

6.4.8. WebODE OWL compliance results

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 (even though these ontologies validate correctly).

The results of a step execution in WebODE, as shown in figure 6.15, 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).
- *Execution fails in the import and export operation.* This occurs in 18 cases (ISA02-05, ISA13-17, ISB08-11, ISE05-06, ISE10, ISG05, ISH02).
- *Execution fails when the ontologies are compared.* This occurs in 9 cases (ISA09-12, ISB04-07, ISE01).

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

Class hierarchies

- **Classes.** An *rdfs:label* property is inserted into the classes with the value “<class_name>”. This occurs whenever classes appear.
- **Named class hierarchies with or without cycles.** When a hierarchy has multiple classes, execution fails. When a class is a subclass of itself, the

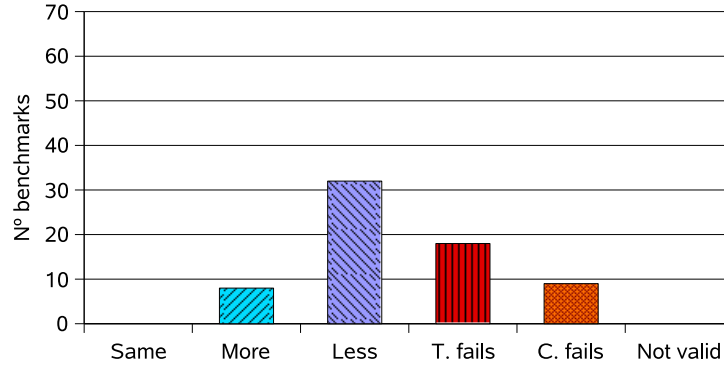


Figure 6.15: OWL import and export operation results for WebODE.

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. A new property is created with an incorrect domain and range²¹ and with a name “<property_name>_1”. The restriction is created with the value constraint *owl:allValuesFrom(owl:Thing)*. In the case of the constraint *owl:someValuesFrom*, the constraint is lost.
- Classes that are a subclass of a cardinality constraint in an object property. The property is created with a domain that is defined as the union of the class and an incorrect name²¹ and with a range that is defined as the union of *owl:Thing* and an incorrect name²¹. 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 constraint *owl:minCardinality*, 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 that are a subclass of a cardinality constraint in a datatype property. Execution fails.
- Classes that are a subclass of a class intersection. Execution fails.

Class equivalences

- Classes equivalent to named classes. The *owl:equivalentClass* property is lost.

²¹ <http://3A-2F-www.w3.org-2F2002-2F07-2Fowl-23Thing>

- Classes equivalent to a value constraint in an object property. 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>_1” and with incorrect domain and range²¹. The anonymous concept is created as a subclass of the restriction and not as equivalent to the restriction. 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 equivalent to a cardinality constraint in an object property. The property is created with a domain that is defined as the union of an anonymous concept and an incorrect name²¹ and with a range that is defined as the union of *owl:Thing* and an incorrect name²¹. 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 constraint *owl:minCardinality*, the constraint in the restriction is lost. In the case of the constraint *owl:maxCardinality*, the value of the constraint is “11” instead of “1”. In the case of the constraint *owl:cardinality*, 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. Execution fails.
- Classes equivalent to a class intersection. The *owl:intersectionOf* and *owl:equivalentClass* properties are lost. An anonymous class is created.

Classes defined with set operators

- Classes that are intersection of other classes. The *owl:intersectionOf* property is lost.

Properties

- Object and datatype properties. An *rdfs:label* is inserted into the properties with the value “<property_name>”. This occurs whenever properties appear.
- Object and datatype property hierarchies. The *rdfs:subPropertyOf* properties are lost.
- Object properties without domain or range. When there are object properties without domain, the domain is created with an incorrect name²¹. When there are object properties without range, the range is created with an incorrect name²¹ and the class is created as a subclass of the restriction *restriction(owl:Thing owl:allValuesFrom(owl:Thing))*.

- Datatype properties without domain or range. 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(a:hasSSN owl:allValuesFrom(xsd:string))* restriction and the range is created as *xsd:string*.
- Object properties with domain and range. The class is created as a subclass of the *restriction(a:hasChild owl:allValuesFrom(a:Person))* restriction.
- Datatype properties with domain and range. The class is created as a subclass of the *restriction(a:hasSSN owl:allValuesFrom(xsd:string))* restriction. The range changes from *rdfs:Literal* to *xsd:string*.
- Object and datatype properties with multiple domains or ranges. Execution fails.

Relations between properties

- Equivalent object and datatype properties. The *owl:equivalentProperty* property is lost.
- Inverse object properties. The *owl:inverseOf* property is lost.

Global cardinality constraints and logical property characteristics

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

Individuals

- Individuals. An *rdfs:label* property is inserted into the individuals with the value “<individual_name>”. This occurs whenever individuals appear.
- Individuals of a single class. The individuals remain the same.
- Individuals of multiple classes. Execution fails.
- Named individuals and object properties. The property with the value in the instance is lost.
- Named individuals and datatype properties. The value in the property is changed from “<value>” to “<value>”^{^xsd:string}.
- Anonymous individuals and object or datatype properties. The anonymous individual is created as a named individual.

Individual identity

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

6.4.9. Global OWL compliance results

Table 6.4 presents the results of a step execution for each tool²². It shows the number of benchmarks in each category in which the results of a step execution can be classified:

- *The original and the resultant ontologies are the same.* The only tools that always produce the same ontologies are Jena, Protégé-OWL and SWI-Prolog. Frame-based tools (Protégé-Frames and WebODE) rarely produce the same ontologies and this is because they usually insert and remove information when importing and exporting.
- *The resultant ontology includes more information than the original one.* This only happens with Protégé-Frames and WebODE, as they insert *rdfs:label* properties to classes and properties with their names.
- *The resultant ontology includes less information than the original one.* In this case, information is sometimes inserted into the resultant ontology.
- *Execution fails in the import and export operation.* The only tool that has execution problems is WebODE.
- *Execution fails when the ontologies are compared.* There are several cases in which the execution of the comparer fails when two ontologies are compared, as we observed in the evaluation of the comparer. This does not allow us to know whether the tool behaves correctly or not, but pinpoints cases that should be analysed in detail. Nevertheless, these numbers are an indicator of the low robustness of the comparer used.

	GA	JE	K2	PF	PO	ST	SP	WE
Same	79	82	63	4	82	39	82	
More				4				16
Less	2		11	56		33		39
Tool fails								18
Comp. fails	1		8	18		10		9

Table 6.4: Results in *Step 1* (for 82 benchmarks).

²²The tool names have been abbreviated in the tables: GA=GATE, JE=Jena, K2=KAON2, PF=Protégé-Frames, PO=Protégé-OWL, ST=SemTalk, SP=SWI-Prolog, and WE=WebODE.

Table 6.5 is a breakdown of the row “*Same*” in table 6.4 according to the combination of components present in the ontology; it shows the percentage of benchmarks in which the original (O_i) and the resultant (O_i^{II}) ontologies in *Step 1* are the same. We can observe that some tools work better with some combinations of components than with others.

Group	GA	JE	K2	PF	PO	ST	SP	WE
A - Class hierarchies	47	100	71	6	100	35	100	
B - Class equivalences	50	100	75		100		100	
C - Classes defined with set operators	50	100	100		100	100	100	
D - Property hierarchies	50	100	50	50	100	75	100	
E - Properties with domain and range	50	100	100		100	70	100	
F - Relations between properties	33	100	100		100	33	100	
G - Global cardinality constraints and logical property characteristics	60	100	100		100	60	100	
H - Single individuals		100	100		100	100	100	
I - Named individuals and properties	40	100	100		100	60	100	
J - Anonymous individuals and properties	67	100	100		100		100	
K - Individual identity	33	100	100		100	33	100	
L - Syntax and abbreviation	53	100	47	53	100	60	100	

Table 6.5: Percentage of identical ontologies per group in *Step 1*.

If we classify the mismatches found when comparing the original (O_i) and the resultant (O_i^{II}) ontologies according to the levels of ontology translation problems described in section 1.3.3, we can see that mismatches are found in all the levels except in the *Lexical* one, being most of them in the *Conceptual* level. We can also observe that, in some cases, mismatches occur in two levels when two ontologies are compared (e.g., in the *Syntactic* and *Conceptual* levels).

Next, we show for each level the tools that present mismatches and the causes of these mismatches:

- *Pragmatic* level. Mismatches found at this level occur when the two ontologies are semantically equivalent but their interpretation may be different in different contexts (e.g., an ontology contains two classes that are subclass of the other, thus forming a cycle, and another ontology contains two classes that are equivalent). The tools with mismatches at this level are Protégé-Frames, Protégé-OWL, SemTalk, and WebODE.
- *Conceptual* level. Mismatches found at this level are due to differences in

conceptualizations and are the most frequent in the tools. The tools with mismatches at this level are GATE, KAON2, Protégé-Frames, SemTalk, and WebODE.

- *Terminological* level. The tools with mismatches at this level are Protégé-Frames, which changes the names of ontologies, classes, properties, instances, and gives name to anonymous individuals; SemTalk, which loses the name of the ontology; and WebODE, which gives name to anonymous individuals.
- *Paradigm* level. The only tool with mismatches at this level is Protégé-Frames when it defines classes as a subclass of *owl:Thing*, because in Protégé-Frames all the classes are subclass of *Thing*.
- *Syntactic* level. The tools with mismatches at the *Syntactic* level are GATE, and Protégé-Frames because they change the ontology into OWL Full, and WebODE because it redefines datatypes.

In GATE, the change to OWL Full occurs because it does not explicitly define classes as *owl:Class*. In Protégé-Frames, the change to OWL Full occurs because it generates RDF properties (*rdf:Property*) instead of OWL datatype properties (*owl:DatatypeProperty*). In WebODE, mismatches are due to the redefinition of datatypes or the insertion of them in values that were not typed.

6.5. OWL interoperability results

With the previous information on the OWL compliance of the tools, this section presents the analysis of the OWL interoperability of the tools that participated in the benchmarking.

First, an overview of the OWL interoperability of the tools from an in-depth study of each participating tool is provided, including some interoperability issues not detected in the analysis of the OWL compliance. Second, an analysis of the OWL interoperability of the tools from a global viewpoint is presented. A detailed analysis of these results can be found in [García-Castro *et al.*, 2007a].

6.5.1. OWL interoperability results per tool

This section presents the OWL interoperability results of each tool; the results of the interoperability between two tools (e.g., T1 and T2) include the interchange from one tool to the same tool (from T1 to T1 – column “ $\Leftrightarrow$ ”), from one tool to another (from T1 to T2 – column “ $\Rightarrow$ ”) and vice versa (from T2 to T1 – column “ $\Leftarrow$ ”).

The interoperability results have been grouped according to the combinations of components present in the benchmarks. As table 6.6 shows, subgroups have been defined from the groups of the OWL Lite Import Benchmark Suite to increase the granularity of the analysis.

Subgroups	Benchmarks
A - Class hierarchies Named class hierarchies without cycles Named class hierarchies with cycles Classes subclass of a value constraint in an object property Classes subclass of a cardinality constraint in an object property Classes subclass of a cardinality constraint in a datatype property Classes subclass of a class intersection	ISA01-ISA04 ISA05-ISA06 ISA07-ISA08 ISA09-ISA12 ISA13-ISA16 ISA17
B - Class equivalences Equivalent named classes Classes equivalent to a value constraint in an object property Classes equivalent to a cardinality constraint in an object property Classes equivalent to a cardinality constraint in a datatype property Classes equivalent to a class intersection	ISB01 ISB02-ISB03 ISB04-ISB07 ISB08-ISB11 ISB12
C - Classes defined with set operators Classes intersection of other classes	ISC01-ISC02
D - Property hierarchies Object property hierarchies Datatype property hierarchies	ISD01-ISD02 ISD03-ISD04
E - Properties with domain and range Object properties without domain or range Object properties with domain and range Object properties with multiple domains or ranges Datatype properties without domain or range Datatype properties with domain and range Datatype properties with multiple domains	ISE01-ISE02 ISE03-ISE04 ISE05-ISE06 ISE07-ISE08 ISE09 ISE10
F - Relations between properties Equivalent object and datatype properties Inverse object properties	ISF01-ISF02 ISF03
G - Global cardinality constraints and logical property characteristics Transitive object properties Symmetric object properties Functional object and datatype properties Inverse functional object properties	ISG01 ISG02 ISG03-ISG04 ISG05
H - Single individuals Instances Instances of multiple classes	ISH01, ISH03 ISH02
I - Named individuals and properties Named individuals and object properties Named individuals and datatype properties	ISI01-ISI03 ISI04-ISI05
J - Anonymous individuals and properties Anonymous individuals and object properties Anonymous individuals and datatype properties	ISJ01-ISJ02 ISJ03
K - Individual identity Equivalent individuals Different individuals	ISK01 ISK02-ISK03

Table 6.6: Subgroups of the OWL import benchmarks.

These results are restrictive, i.e., when a single benchmark in a subgroup has any problem in one of the directions of the interchange, the whole subgroup has this problem. The results of any subgroup can be the following:

- **SAME** (*S*). When all the ontologies interchanged between two tools are the same (all the benchmarks in the subgroup have an *INTEROPERABILITY* result of *SAME*).
- **DIFFERENT** (*D*). When at least one ontology interchanged between two tools is different and no execution errors exist (any benchmark in the subgroup 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 subgroup has an *EXECUTION* result of *N.E.* – Non Executed).

Tables 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13, and 6.14 show a summary of the interoperability results of GATE, Jena, KAON2, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, and WebODE with the other tools, respectively.

The results of the interoperability of the tools participating in the benchmarking depend not just on their behaviour during the import and export operation (as described in section 6.4) 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 so because sometimes 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, the tools 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, if the ontologies include classes with multiple instances, then 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, SemTalk 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, GATE produces ontologies that make the comparer execution fail.

Subgroups	GA-GA	GA-JE	GA-K2	GA-PF	GA-PO	GA-ST	GA-SP	GA-WE
Class hierarchies								
Named class hierarchies without cycles	-	S	D	D	-	D	S	-
Named class hierarchies with cycles	-	S	S	D	-	D	-	-
Classes subclass of a value constraint in an object property	-	S	S	D	-	D	S	-
Classes subclass of a cardinality constraint in an object property	-	S	-	D	-	-	S	-
Classes subclass of a cardinality constraint in a datatype property	-	S	-	D	-	-	-	-
Classes subclass of a class intersection	-	S	S	D	-	S	S	-
Class equivalences								
Equivalent named classes	S	S	S	D	S	D	S	-
Classes equivalent to a value constraint in an object property	-	S	S	-	-	-	-	-
Classes equivalent to a cardinality constraint in an object property	-	S	-	D	S	-	-	-
Classes equivalent to a cardinality constraint in a datatype property	-	S	-	-	-	-	-	-
Classes equivalent to a class intersection	S	S	S	D	S	D	S	-
Classes defined with set operators								
Classes intersection of other classes	-	S	S	D	S	-	S	-
Property hierarchies								
Object property hierarchies	-	S	D	D	S	D	S	-
Datatype property hierarchies	D	S	D	D	-	D	-	-
Properties with domain and range								
Object properties without domain or range	-	S	S	D	-	D	-	-
Object properties with domain and range	-	S	S	D	-	D	-	-
Object properties with multiple domains or ranges	-	S	-	D	-	D	-	-
Datatype properties without domain or range	-	S	S	D	-	D	-	-
Datatype properties with domain and range	-	S	S	D	-	D	-	-
Datatype properties with multiple domains	S	S	S	D	-	D	S	-
Relations between properties								
Equivalent object and datatype properties	-	S	S	D	-	D	S	-
Inverse object properties	-	S	S	D	-	D	S	-
Global cardinality constraints and logical property characteristics								
Transitive object properties	S	S	S	D	-	D	S	-
Symmetric object properties	-	S	S	D	-	D	S	-
Functional object properties	-	S	S	D	-	D	S	-
Inverse functional object properties	S	S	S	D	-	D	S	-
Single individuals								
Instances	-	-	-	-	-	-	-	-
Instances of multiple classes	D	S	D	-	D	-	D	-
Named individuals and properties								
Named individuals and object properties	-	S	S	D	-	D	-	-
Named individuals and datatype properties	-	S	S	-	-	-	-	-
Anonymous individuals and properties								
Anonymous individuals and object properties	-	D	S	D	-	-	-	-
Anonymous individuals and datatype properties	S	D	D	-	-	-	-	-
Individual identity								
Equivalent individuals	-	S	-	D	S	D	S	-
Different individuals	-	S	S	-	S	-	S	-
Syntax and abbreviation								
Syntax and abbreviation	-	S	D	-	-	-	-	-

Table 6.7: OWL interoperability results of GATE.

Subgroups	JE-GA		JE-JE		JE-K2		JE-PF		JE-PO		JE-ST		JE-SP		JE-WE	
	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐
Class hierarchies																
Named class hierarchies without cycles	S	S	S	S	D	D	D	D	S	S	D	S	S	S	-	-
Named class hierarchies with cycles	S	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Classes subclass of a value constraint in an object property	S	S	S	S	S	S	D	D	S	S	-	D	S	S	D	D
Classes subclass of a cardinality constraint in an object property	-	S	S	S	-	-	D	D	S	S	-	-	S	S	-	-
Classes subclass of a cardinality constraint in a datatype property	S	S	S	S	S	S	-	D	S	S	-	S	S	S	-	-
Classes subclass of a class intersection	S	S	S	S	S	S	D	D	S	S	-	S	S	S	-	-
Class equivalences																
Equivalent named classes	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Classes equivalent to a value constraint in an object property	S	S	S	S	S	S	D	D	S	S	-	D	S	S	-	-
Classes equivalent to a cardinality constraint in an object property	-	S	S	S	-	-	D	D	S	S	-	-	S	S	-	-
Classes equivalent to a cardinality constraint in a datatype property	S	S	S	S	S	S	-	D	S	S	-	D	S	S	-	D
Classes equivalent to a class intersection	S	S	S	S	S	S	D	D	S	S	-	D	S	S	D	D
Classes defined with set operators																
Classes intersection of other classes	S	S	S	S	S	S	D	D	S	S	-	S	S	S	D	D
Property hierarchies																
Object property hierarchies	D	S	S	S	D	D	D	D	S	S	D	S	S	S	D	D
Datatype property hierarchies	D	S	S	S	D	D	D	D	S	S	D	S	S	S	D	D
Properties with domain and range																
Object properties without domain or range	S	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Object properties with domain and range	S	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Object properties with multiple domains or ranges	-	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Datatype properties without domain or range	S	S	S	S	S	S	-	D	S	S	D	S	S	S	D	D
Datatype properties with domain and range	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Datatype properties with multiple domains	S	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Relations between properties																
Equivalent object and datatype properties	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Inverse object properties	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Global cardinality constraints and logical property characteristics																
Transitive object properties	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Symmetric object properties	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Functional object and datatype properties	S	S	S	S	S	S	D	D	S	S	-	D	S	S	-	-
Inverse functional object properties	S	S	S	S	S	S	D	D	S	S	-	D	S	S	-	-
Single individuals																
Instances	-	-	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Instances of multiple classes	S	S	S	S	S	S	D	D	S	S	D	S	S	S	-	-
Named individuals and properties																
Named individuals and object properties	S	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Named individuals and datatype properties	S	S	S	S	S	S	-	-	S	S	D	S	S	S	D	D
Anonymous individuals and properties																
Anonymous individuals and object properties	-	D	S	S	S	S	D	D	S	S	-	D	S	S	D	D
Anonymous individuals and datatype properties	D	D	S	S	S	S	-	-	S	S	-	D	S	S	D	D
Individual identity																
Equivalent individuals	-	S	S	S	S	S	D	D	S	S	D	S	S	S	D	D
Different individuals	D	S	S	S	S	S	D	D	S	S	-	D	S	S	D	D
Syntax and abbreviation																
Syntax and abbreviation	S	S	S	S	D	D	-	-	S	S	-	-	S	S	D	D

Table 6.8: OWL interoperability results of Jena.

Subgroups	K2-GA	K2-JE	K2-K2	K2-PF	K2-PO	K2-ST	K2-SP	K2-WE
Class hierarchies								
Named class hierarchies without cycles	-	D	D	D	D	D	D	-
Named class hierarchies with cycles	-	S	S	D	S	D	S	-
Classes subclass of a value constraint in an object property	-	S	S	D	S	D	S	D
Classes subclass of a cardinality constraint in an object property	-	-	-	-	-	-	-	-
Classes subclass of a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-
Classes subclass of a class intersection	S	S	S	D	S	S	S	-
Class equivalences								
Equivalent named classes	-	S	S	D	S	S	S	D
Classes equivalent to a value constraint in an object property	-	S	S	D	S	D	S	D
Classes equivalent to a cardinality constraint in an object property	-	-	-	-	-	-	-	-
Classes equivalent to a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-
Classes equivalent to a class intersection	S	S	S	D	S	D	S	D
Classes defined with set operators								
Classes intersection of other classes	-	S	S	D	S	-	S	D
Property hierarchies								
Object property hierarchies	-	D	D	D	D	D	D	D
Datatype property hierarchies	-	D	D	D	D	D	D	D
Properties with domain and range								
Object properties without domain or range	-	S	S	D	S	S	S	-
Object properties with domain and range	-	S	S	D	S	S	S	-
Object properties with multiple domains or ranges	-	S	S	D	S	S	S	-
Datatype properties without domain or range	-	S	S	-	S	D	S	-
Datatype properties with domain and range	S	S	S	D	S	D	S	-
Datatype properties with multiple domains	S	S	S	D	S	D	S	-
Relations between properties								
Equivalent object and datatype properties	-	S	S	D	S	D	S	D
Inverse object properties	-	S	S	D	S	D	S	D
Global cardinality constraints and logical property characteristics								
Transitive object properties	S	S	S	D	S	S	S	D
Symmetrical object properties	-	S	S	D	S	S	S	D
Functional object and datatype properties	S	S	S	D	S	D	S	D
Inverse functional object properties	-	-	-	-	-	-	-	-
Single individuals								
Instances	-	-	S	D	S	S	S	-
Instances of multiple classes	-	D	S	D	S	S	S	-
Named individuals and properties								
Named individuals and object properties	-	S	S	D	S	S	S	D
Named individuals and datatype properties	-	S	S	-	S	S	S	D
Anonymous individuals and properties								
Anonymous individuals and object properties	-	S	S	D	S	D	S	D
Anonymous individuals and datatype properties	S	D	S	-	S	D	S	D
Individual identity								
Equivalent individuals	-	S	S	D	S	S	S	D
Different individuals	-	S	S	D	S	D	S	-
Syntax and abbreviation								
Syntax and abbreviation	-	D	D	-	D	D	D	D

Table 6.9: OWL interoperability results of KAON2.

Subgroups	PF-GA		PF-JE		PF-K2		PF-PF		PF-PO		PF-ST		PF-SP		PF-WE	
	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐
Class hierarchies																
Named class hierarchies without cycles	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Named class hierarchies with cycles	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes subclass of a value constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes subclass of a cardinality constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes subclass of a cardinality constraint in a datatype property	-	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a class intersection	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Class equivalences																
Equivalent named classes	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes equivalent to a value constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes equivalent to a cardinality constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes equivalent to a cardinality constraint in a datatype property	-	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a class intersection	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Classes defined with set operators																
Classes intersection of other classes	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Property hierarchies																
Object property hierarchies	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Datatype property hierarchies	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Properties with domain and range																
Object properties without domain or range	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Object properties with domain and range	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Object properties with multiple domains or ranges	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Datatype properties without domain or range	-	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Datatype properties with domain and range	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Datatype properties with multiple domains	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Relations between properties																
Equivalent object and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Inverse object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Global cardinality constraints and logical property characteristics																
Transitive object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Symmetric object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Functional object and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Inverse functional object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Single individuals																
Instances	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Instances of multiple classes	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-	D
Named individuals and properties																
Named individuals and object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Named individuals and datatype properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Anonymous individuals and properties																
Anonymous individuals and object properties	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Anonymous individuals and datatype properties	-	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Individual identity																
Equivalent individuals	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Different individuals	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-	D
Syntax and abbreviation																
Syntax and abbreviation	-	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.10: OWL interoperability results of Protégé-Frames.

Subgroups	PO-GA		PO-JE		PO-K2		PO-PF		PO-PO		PO-ST		PO-SP		PO-WE	
	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐
Class hierarchies																
Named class hierarchies without cycles	D	-	S	S	D	D	D	D	S	D	S	S	S	S	-	-
Named class hierarchies with cycles	S	-	S	S	S	S	D	D	S	D	D	D	S	S	-	-
Classes subclass of a value constraint in an object property	S	-	S	S	S	-	D	D	S	-	-	-	S	S	D	D
Classes subclass of a cardinality constraint in an object property	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Classes subclass of a cardinality constraint in a datatype property	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Classes subclass of a class intersection	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Class equivalences																
Equivalent named classes	S	S	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Classes equivalent to a value constraint in an object property	S	-	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Classes equivalent to a cardinality constraint in an object property	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Classes equivalent to a cardinality constraint in a datatype property	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Classes equivalent to a class intersection	S	-	S	S	S	-	D	D	S	-	-	-	S	S	-	-
Classes defined with set operators																
Classes intersection of other classes	S	S	S	S	S	S	D	D	S	D	D	S	S	S	D	D
Property hierarchies																
Object property hierarchies	D	S	S	S	D	D	D	D	S	D	S	S	S	S	D	D
Datatype property hierarchies	D	-	S	S	D	D	D	D	S	D	S	S	S	S	D	D
Properties with domain and range																
Object properties without domain or range	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Object properties with domain and range	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Object properties with multiple domains or ranges	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Datatype properties without domain or range	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Datatype properties with domain and range	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Datatype properties with multiple domains	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Relations between properties																
Equivalent object and datatype properties	S	-	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Inverse object properties	D	-	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Global cardinality constraints and logical property characteristics																
Transitive object properties	S	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Symmetric object properties	S	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Functional object and datatype properties	S	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Inverse functional object properties	S	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Single individuals																
Instances of multiple classes	S	-	S	S	S	S	D	D	S	D	S	S	S	S	-	-
Named individuals and properties																
Named individuals and object properties	D	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Named individuals and datatype properties	D	-	S	S	S	S	D	D	S	D	S	S	S	S	D	D
Anonymous individuals and properties																
Anonymous individuals and object properties	D	-	S	S	S	S	D	D	S	-	-	D	S	S	D	D
Anonymous individuals and datatype properties	D	-	S	S	S	S	D	D	S	-	-	D	S	S	D	D
Individual identity																
Equivalent individuals	S	S	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Different individuals	S	S	S	S	S	S	D	D	S	D	D	D	S	S	D	D
Syntax and abbreviation																
Syntax and abbreviation	S	-	S	S	D	D	-	-	S	-	-	-	-	S	D	D

Table 6.11: OWL interoperability results of Protégé-OWL.

Subgroups	ST-GA		ST-JE		ST-K2		ST-PF		ST-PO		ST-ST		ST-SP		ST-WE	
	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐
Class hierarchies																
Named class hierarchies without cycles	-	D	S	D	D	D	D	-	S	D	S	D	S	S	-	-
Named class hierarchies with cycles	D	D	D	D	D	D	D	-	D	D	D	D	D	D	-	-
Classes subclass of a value constraint in an object property	-	-	D	-	-	D	-	-	D	-	D	D	D	D	-	D
Classes subclass of a cardinality constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a class intersection	S	-	S	-	-	S	-	-	S	-	S	-	S	S	-	-
Class equivalences																
Equivalent named classes	D	D	D	D	D	S	D	-	D	D	D	D	D	D	D	D
Classes equivalent to a value constraint in an object property	-	-	D	-	-	D	-	-	D	-	D	D	D	D	-	-
Classes equivalent to a cardinality constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a class intersection	D	-	D	-	D	S	D	-	D	-	D	D	D	D	-	D
Classes defined with set operators																
Classes intersection of other classes	-	-	S	-	-	S	-	-	S	-	S	-	S	S	-	-
Property hierarchies																
Object property hierarchies	-	D	S	D	D	D	D	-	S	D	S	D	S	S	D	D
Datatype property hierarchies	D	D	S	D	D	D	D	-	S	D	S	D	S	S	D	D
Properties with domain and range																
Object properties without domain or range	-	D	S	D	S	S	D	-	S	D	S	D	S	S	-	-
Object properties with domain and range	-	D	S	D	S	S	D	-	S	D	S	D	S	S	-	-
Object properties with multiple domains or ranges	-	D	S	D	S	S	D	-	S	D	S	D	S	S	-	-
Datatype properties without domain or range	D	D	D	D	D	D	D	-	D	D	D	D	D	D	D	D
Datatype properties with domain and range	-	D	D	D	D	D	-	-	D	D	D	D	D	D	D	D
Datatype properties with multiple domains	D	D	D	D	D	D	-	-	D	D	D	D	D	D	-	-
Relations between properties																
Equivalent object and datatype properties	-	D	D	D	D	D	-	-	D	D	D	D	D	D	D	D
Inverse object properties	-	D	D	D	D	D	-	-	D	D	D	D	D	D	D	D
Inverse datatype properties	-	D	D	D	D	D	-	-	D	D	D	D	D	D	D	D
Global cardinality constraints and logical property characteristics																
Transitive object properties	S	D	S	D	S	S	D	-	S	D	S	D	S	S	D	D
Symmetric object properties	-	D	S	D	D	S	D	-	S	D	S	D	S	S	D	D
Functional object and datatype properties	-	D	D	-	D	D	-	-	D	D	D	D	D	D	D	D
Inverse functional object properties	D	D	D	D	D	D	-	-	D	D	D	D	D	D	D	-
Single individuals																
Instances	-	-	S	D	S	S	D	-	S	D	S	D	S	S	-	-
Instances of multiple classes	-	-	S	D	S	S	D	-	S	D	S	D	S	S	-	-
Named individuals and properties																
Named individuals and object properties	-	D	S	D	S	S	D	-	S	D	S	D	S	S	D	D
Named individuals and datatype properties	-	D	D	D	D	D	-	-	D	D	D	D	D	D	D	D
Anonymous individuals and properties																
Anonymous individuals and object properties	-	-	D	-	D	S	D	-	D	-	D	D	D	D	D	D
Anonymous individuals and datatype properties	-	-	D	-	D	D	-	-	D	-	D	D	D	D	D	D
Individual identity																
Equivalent individuals	D	D	D	D	D	S	D	-	D	D	D	D	D	D	D	D
Different individuals	-	-	D	-	-	D	-	-	D	-	D	-	D	D	-	-
Syntax and abbreviation																
Syntax and abbreviation	-	-	-	-	-	D	-	-	-	-	-	-	-	-	-	-

Table 6.12: OWL interoperability results of SemTalk.

Subgroups	SP-GA	SP-JE	SP-K2	SP-PF	SP-PO	SP-ST	SP-SP	SP-WE
Class hierarchies								
Named class hierarchies without cycles	-	S	D	D	S	S	S	-
Named class hierarchies with cycles	-	S	S	D	S	D	S	-
Classes subclass of a value constraint in an object property	-	S	S	D	D	D	S	D
Classes subclass of a cardinality constraint in an object property	-	S	-	D	-	-	S	-
Classes subclass of a cardinality constraint in a datatype property	-	S	-	D	-	-	S	-
Classes subclass of a class intersection	S	S	S	-	S	S	S	-
Class equivalences								
Equivalent named classes	S	S	S	D	S	D	S	D
Classes equivalent to a value constraint in an object property	-	S	S	D	S	D	S	-
Classes equivalent to a cardinality constraint in an object property	-	S	S	D	S	-	S	-
Classes equivalent to a cardinality constraint in a datatype property	-	S	-	D	-	-	S	-
Classes equivalent to a class intersection	-	S	S	-	S	D	S	D
Classes defined with set operators								
Classes intersection of other classes	S	S	S	D	S	S	S	D
Property hierarchies								
Object property hierarchies	-	S	D	D	S	S	S	D
Datatype property hierarchies	-	S	D	D	S	D	S	D
Properties with domain and range								
Object properties without domain or range	-	S	S	D	S	S	S	-
Object properties with domain and range	-	S	S	D	S	S	S	D
Object properties with multiple domains or ranges	S	S	S	-	D	D	S	-
Datatype properties without domain or range	-	S	S	D	S	D	S	-
Datatype properties with domain and range	S	S	S	D	S	D	S	-
Datatype properties with multiple domains	S	S	S	-	S	D	S	-
Relations between properties								
Equivalent object and datatype properties	S	S	S	D	S	D	S	D
Inverse object properties	S	S	S	D	S	D	S	D
Global cardinality constraints and logical property characteristics								
Transitive object properties	-	S	S	D	S	S	S	D
Symmetrical object properties	S	S	S	D	S	S	S	D
Functional object and datatype properties	-	S	S	D	S	D	S	-
Inverse functional object properties	S	S	S	D	S	D	S	-
Single individuals								
Instances	-	S	S	D	S	S	S	-
Instances of multiple classes	-	D	S	-	S	S	S	-
Named individuals and properties								
Named individuals and object properties	-	-	S	D	S	S	S	D
Named individuals and datatype properties	-	-	S	-	S	D	S	-
Anonymous individuals and properties								
Anonymous individuals and object properties	-	-	S	D	S	D	S	D
Anonymous individuals and datatype properties	-	-	S	-	S	D	S	-
Individual identity								
Equivalent individuals	S	S	S	D	S	D	S	D
Different individuals	-	S	S	-	S	D	S	-
Syntax and abbreviation								
Syntax and abbreviation	-	-	D	-	-	-	S	-

Table 6.13: OWL interoperability results of SWI-Prolog.

Subgroups	WE-GA		WE-JE		WE-K2		WE-PF		WE-PO		WE-ST		WE-SP		WE-WE	
	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$	$\Rightarrow$	$\Leftarrow$
Class hierarchies																
Named class hierarchies without cycles	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Named class hierarchies with cycles	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Names subclass of a value constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a cardinality constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes subclass of a class intersection	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Class equivalences																
Equivalent named classes	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a value constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a cardinality constraint in an object property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a cardinality constraint in a datatype property	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes equivalent to a class intersection	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Classes defined with set operators																
Classes intersection of other classes	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Property hierarchies																
Object property hierarchies	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Datatype property hierarchies	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Properties with domain and range																
Object properties without domain or range	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Object properties with domain and range	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Object properties with multiple domains or ranges	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Datatype properties without domain or range	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Datatype properties with domain and range	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Datatype properties with multiple domains	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Relations between properties																
Equivalent object and datatype properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Inverse object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Global cardinality constraints and logical property characteristics																
Transitive object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Symmetric object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Functional object and datatype properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Inverse functional object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Single individuals																
Instances	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Instances of multiple classes	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Named individuals and properties																
Named individuals and object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Named individuals and datatype properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Anonymous individuals and properties																
Anonymous individuals and object properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Anonymous individuals and datatype properties	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Individual identity																
Equivalent individuals	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Different individuals	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Syntax and abbreviation																
Syntax and abbreviation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.14: OWL interoperability results of WebODE.

- In the case of interchanges **from SemTalk to Jena**, when Jena uses ontologies generated by SemTalk, Jena loses the datatype property hierarchies.
- In the case of interchanges **from SemTalk to KAON2**, when KAON2 uses ontologies generated by SemTalk, KAON2 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, SemTalk loses the *rdfs:subClassOf* and *rdfs:subPropertyOf* properties; it loses the domain and the range in object or datatype properties with domain or range; and its execution fails with a) classes that are a subclass or equivalent to value constraints, cardinality constraints, or class intersections, b) classes that are intersection of other classes, and c) anonymous individuals with object or datatype properties.
- In the case of interchanges **from Protégé-Frames, Protégé-OWL and WebODE to SWI-Prolog**, when SWI-Prolog uses all the ontologies produced by Protégé-Frames, most of the ontologies produced by WebODE, and some of the ontologies produced by Protégé-OWL, SWI-Prolog generates ontologies that are not valid in the RDF/XML syntax. SWI-Prolog produces ontologies with an incorrect namespace identifier (*[]*) when it imports ontologies that contain default namespaces (*xmlns="namespaceURI"*).

6.5.2. Global OWL interoperability results

Table 6.15 gives an overview of the interoperability between the tools and shows the percentage of benchmarks in which the original (O_i) and the resultant (O_i^{IV}) ontologies in an interchange are the same. For each cell, the row indicates the tool origin of the interchange, whereas the column indicates the tool destination of the interchange.

		DESTINATION							
		JE	PO	SP	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	78	85	16	5	
	PO	100	100	95	78	89	16	5	
	SP	100	100	100	78	55	45	5	
	KA	78	78	78	78	40	39		
	GA	96	52	79	74	46	13	13	
	ST	45	46	46	27	24	46		
	PF	5	5			4	5	13	
	WE							11	

Table 6.15: Percentage of identical interchanged ontologies.

The first thing that can be glanced at is that the interoperability between the tools is low, even in interchanges between a tool and the tool itself.

It is also clear from the results that interoperability using OWL as interchange language depends on the knowledge model of the tools, and that the more similar the knowledge model of a tool is to OWL, the more interoperable the tool is. Nevertheless, the way of serializing the ontologies in the RDF/XML syntax also has a high influence in the results, as the interoperability issues identified above show.

A correct working of tool importers and exporters does not ensure interoperability. In section 6.4 we saw that Jena, Protégé-OWL and SWI-Prolog always produced the same ontologies in *Step 1*, but not all of these tools also produce the same ontologies after interchanging them. Interchanges between Jena and Protégé-OWL and interchanges between Jena and SWI-Prolog do produce the same ontologies. In the interchanges between Protégé-OWL and SWI-Prolog, when the interchange is from SWI-Prolog to Protégé-OWL the ontologies produced are the same, but when the interchange is from Protégé-OWL to SWI-Prolog some problems arise²³.

This leads us to a second fact, that interoperability between two tools is usually different depending on the direction of the interchange. This can be clearly observed in the table and in the previous example about Protégé-OWL and SWI-Prolog.

To analyse the **interoperability of the tools regarding the combination of components present in the ontology** and to know in which cases interchanges can be performed in one direction but not in both, we have analysed for each group of the OWL Lite Import Benchmark Suite the percentage of benchmarks in which the original (O_i) and the resultant (O_i^{IV}) ontologies in an interchange are the same. Tables 6.16 to 6.26 provide these results.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	71	88	6		
	PO	100	100	100	71	88	6		
	SW	100	100	100	71	53	35		
	KA	71	71	71	71	24	29		
	GA	100	29	100	71	47	6	6	
	ST	35	35	35	18	18	35		
	PF							6	
	WE							6	

Table 6.16: Percentage of identical interchanged ontologies for *Class hierarchies*.

Tables 6.27 and 6.28 provide an overview of the **robustness of the tools** and show the percentage of benchmarks in which tool execution fails in the first and in the second step, respectively.

We can see that, with the exception of WebODE, tools have no execution problems when processing the ontologies of the benchmark suite but some of

²³SWI-Prolog produces ontologies with an incorrect namespace identifier (`[/]`) when it imports ontologies that contain default namespaces (`xmlns="namespaceURI"`)

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	75	83			
	PO	100	100	100	75	83			
	SW	100	100	100	75	50			
	KA	75	75	75	75	8	17		
	GA	100	92	100	75	50			
	ST								
	PF								
	WE								

Table 6.17: Percentage of identical interchanged ontologies for *Class equivalences*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	100			
	PO	100	100	100	100	100			
	SW	100	100	100	100	50	100		
	KA	100	100	100	100	50	100		
	GA	100	100	100	100	50			
	ST	100	100	100		50	100		
	PF								
	WE								

Table 6.18: Percentage of identical interchanged ontologies for *Classes defined with set operators*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	50	50	50		
	PO	100	100	100	50	50	50		
	SW	100	100	100	50	50	75		
	KA	50	50	50	50	50	25		
	GA	100	75	100	50	50	50	50	
	ST	50	75	75	25	50	75		
	PF							50	
	WE								

Table 6.19: Percentage of identical interchanged ontologies for *Property hierarchies*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	90			
	PO	100	100	100	100	100			
	SW	100	100	100	100	70	70		
	KA	100	100	100	100	60	70		
	GA	100	0	100	100	50			
	ST	70	70	70	70	40	70		
	PF								
	WE								

Table 6.20: Percentage of identical interchanged ontologies for *Properties with domain and range*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	100			
	PO	100	100	100	100	67			
	SW	100	100	100	100	100	33		
	KA	100	100	100	100	33	33		
	GA	100	33	100	100	33			
	ST	33	33	33	33		33		
	PF								
	WE								

Table 6.21: Percentage of identical interchanged ontologies for *Relations between properties*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	100			
	PO	100	100	100	100	100			
	SW	100	100	100	100	60	40		
	KA	100	100	100	100	60	60		
	GA	100	40	100	100	60			
	ST	60	60	60	60	40	60		
	PF								
	WE								

Table 6.22: Percentage of identical interchanged ontologies for *Global cardinality constraints and logical property characteristics*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	67	67		
	PO	100	100	100	100	100	67		
	SW	100	100	100	100	33	100		
	KA	100	100	100	100	33	100		
	GA	67							
	ST	100	100	100	100	33	100		
	PF								
	WE								

Table 6.23: Percentage of identical interchanged ontologies for *Single individuals*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	100			
	PO	100	100	100	100	80			
	SW	100	100	100	100	60	60		
	KA	100	100	100	100	60	60		
	GA	100	60		100	40			
	ST	60	60	60	60	20	60		
	PF								
	WE								

Table 6.24: Percentage of identical interchanged ontologies for *Named individuals and properties*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	33			
	PO	100	100	100	100	33			
	SW	100	100	100	100				
	KA	100	100	100	100	67	67		
	GA	33	33		67	67			
	ST								
	PF								
	WE								

Table 6.25: Percentage of identical interchanged ontologies for *Anonymous individuals and properties*.

		DESTINATION							
		JE	PO	SW	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	100	33			
	PO	100	100	100	100	100			
	SW	100	100	100	100	33	33		
	KA	100	100	100	100	33	67		
	GA	100	100	100	100	33			
	ST	33	33	33			33		
	PF								
	WE								

Table 6.26: Percentage of identical interchanged ontologies for *Individual identity*.

		DESTINATION							
		GA	JE	K2	PF	PO	ST	SP	WE
ORIGIN	GA								
	JE								
	K2								
	PF								
	PO								
	ST								
	SP								
	WE	22	22	22	22	22	22	22	22

Table 6.27: Percentage of benchmarks in which tool execution fails in *Step 1*.

		DESTINATION							
		GA	JE	K2	PF	PO	ST	SP	WE
ORIGIN	GA				1	1	37	17	89
	JE						37		22
	K2								13
	PF						11		
	PO						37		22
	ST								22
	SP								22
	WE	67							16

Table 6.28: Percentage of benchmarks in which tool execution fails in *Step 2*.

them do have problems when processing ontologies generated by other tools. Needless to say, this lack of robustness in the tools also has a negative effect in interoperability.

If we classify the mismatches found when comparing the original (O_i) and the final (O_i^{IV}) ontologies according to the levels of ontology translation problems described in section 1.3.3, we can draw the same conclusions as those for the

OWL compliance of the tools presented in section 6.4.9. The only comment we can add is that mismatches in the first step cause further mismatches in the second step and even tool failure, as the interoperability and robustness results above described illustrate.

Finally, we have identified the **clusters of interoperable tools** according to the OWL interoperability results. Table 6.29 shows the percentage of benchmarks in which the original (O_i) and the resultant (O_i^{IV}) ontologies in an interchange are the same, according to the combination of components present in the ontology. Each column shows the average of the percentages for every tool in the cluster and in all directions²⁴. From left to right, the table shows a cluster with all the tools and successive clusters which have removed the tool with lower percentages from the previous cluster.

In the table, we can see that Jena, KAON2, Protégé-OWL, and SWI-Prolog interchange correctly all the combinations of components except class equivalences and class and property hierarchies. Jena, Protégé-OWL, and SWI-Prolog can interchange correctly all the component combinations; however, since some problems appear when Protégé-OWL interchanges ontologies with SWI-Prolog in the *Syntax and abbreviation* benchmarks, the only two clusters of fully-interoperable tools are Jena with Protégé-OWL and Jena with SWI-Prolog.

6.6. Evolution of OWL interoperability results

As mentioned in chapter 4, once the benchmarking report has been written and its results disseminated, each organization must plan the changes needed to improve their tools and implement these changes.

This section presents an example of the improvement of the OWL interoperability results after debugging one of the tools participating in the benchmarking; that is, we show how the improvement of WebODE (the tool developed in the Ontology Engineering Group, where the candidate works) also entails the improvement of the interoperability of this tool with the others.

In order to clearly identify the improvement (or loss) of interoperability gained after updating WebODE, we chose to perform the experiments maintaining the versions of the tools used in the previous experiments.

We expected to remove all the execution failures of the tool and to correctly interchange with the other tools the common parts of their knowledge models. It must be noted that this second improvement also depends on the other tools participating in the interchanges.

The mechanisms provided to the WebODE developers to measure and monitor the interoperability of the tool were the IBSE tool and the OWL Lite Import Benchmark Suite.

Next, we present the results of WebODE after performing the changes in the tool (version 2.0 build 192), first the OWL compliance results and, second, the OWL interoperability results with all the tools (including WebODE itself).

²⁴i.e., For a cluster of two tools, A and B, we considered: A to B, B to A, A to A, and B to B.

Group	All	GA, JE, K2, PF, PO, ST, SP	GA, JE, K2, PO, ST, SP	GA, JE, K2, PO, SP	JE, K2, PO, SP	JE, PO, SP	JE, SP
Class hierarchies	34	44	58	74	88	100	100
Class equivalences	32	42	56	78	90	100	100
Classes defined with set operators	43	55	74	90	100	100	100
Property hierarchies	38	49	63	68	80	100	100
Properties with domain and range	41	53	70	85	100	100	100
Relations between properties	39	50	67	87	100	100	100
Global cardinality constraints and logical property characteristics	41	53	70	88	100	100	100
Single individuals	40	51	68	67	100	100	100
Named individuals and properties	39	51	68	83	100	100	100
Anonymous individuals and properties	33	43	57	80	100	100	100
Individual identity	36	46	62	82	100	100	100
Syntax and abbreviation	40	51	63	71	76	96	100

Table 6.29: Percentage of identical interchanged ontologies per group.

The result analysis has been made as was made for the results described in the previous section.

6.6.1. OWL compliance results

Here we describe the OWL compliance of the updated version of WebODE, i.e., how it behaves in the combined operation of importing one OWL ontology and exporting it again (a step of the experiment, as defined in section 6.1).

Table 6.30 presents the results of a step execution for WebODE before and after the changes; it shows the number of benchmarks in each category in which the results of a step execution can be classified. Figure 6.16 shows the results of the updated WebODE.

	WebODE 2.0 b140	WebODE 2.0 b192
Same		14
More	16	11
Less	39	57
Tool fails	18	
Comparer fails	9	

Table 6.30: Updated results in *Step 1* (for 82 benchmarks).

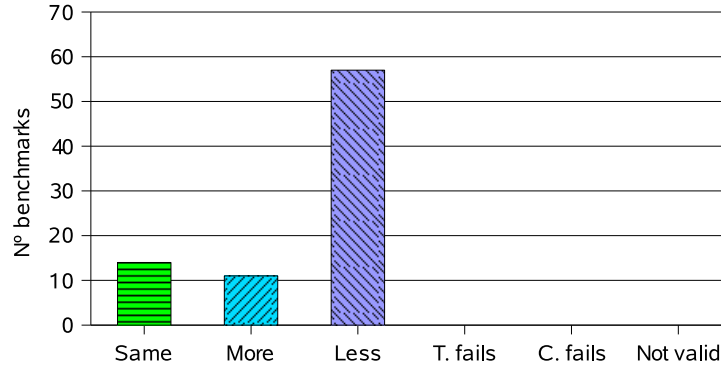


Figure 6.16: Updated OWL import and export operation results for WebODE.

In such results we can observe that the updated version of WebODE does not have execution problems, being more robust, and that WebODE does not make the ontology comparer fail. Furthermore, in some cases, the ontology resulting from the import and export operation is the same than the original one.

The results of a step execution in WebODE, as shown in figure 6.16, can be classified into three categories:

- *The original and the resultant ontologies are the same.* This occurs in 14 cases (ISA01-04, ISH01, ISH03, ISL01-06, ISL09-10).

- *The resultant ontology includes more information than the original one.* This occurs in 11 cases (ISA10, ISD01, ISE01-04, ISG01-02, ISI01-03).
- *The resultant ontology includes less information than the original one.* In this case, information is sometimes inserted into the resultant ontology. This occurs in 57 cases (ISA05-09, ISA11-17, ISB01-12, ISC01-02, ISD02-04, ISE05-10, ISF01-03, ISG03-05, ISH02, ISI04-05, ISJ01-03, ISK01-03, ISL07-08, ISL11-15).

Table 6.31 is a breakdown of the row “*Same*” in table 6.30 according to the combination of components present in the ontology and for the updated WebODE; it contains the percentage of benchmarks in which the original (O_i) and the resultant (O_i^{II}) ontologies in *Step 1* are the same. The table shows that the cases where the original and the resultant ontologies in *Step 1* are the same belong to the *Class hierarchies*, *Single individuals*, and *Syntax and abbreviation* groups; in the case of the latter, this is so because it contains ontologies from the previous groups.

Group	WebODE 2.0 b192
A - Class hierarchies	24
B - Class equivalences	
C - Classes defined with set operators	
D - Property hierarchies	
E - Properties with domain and range	
F - Relations between properties	
G - Global cardinality constraints and logical property characteristics	
H - Single individuals	67
I - Named individuals and properties	
J - Anonymous individuals and properties	
K - Individual identity	
L - Syntax and abbreviation	53

Table 6.31: Updated percentage of identical ontologies per group in *Step 1*.

If we classify the mismatches found when comparing the original and the resultant ontologies according to the levels of ontology translation problems described in section 1.3.3, we see that we can find mismatches in all but in the *Syntactic* and the *Lexical* levels, being most of them in the *Conceptual* level; in some cases, however, mismatches occur in two levels (e.g., in the *Paradigm* and *Conceptual* levels).

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

Class hierarchies

- Named class hierarchies without cycles. The ontologies processed remain the same.
- Named class hierarchies with cycles. One *rdfs:subClassOf* property is removed to eliminate the cycle. When a class is a subclass of itself, the ontology processed is different but semantically the same.
- Classes that are a subclass of a value constraint in an object property. The property is created with domain the class and range *owl:Thing*. The restriction is created with the value constraint *owl:allValuesFrom(owl:Thing)*. In the case of the constraint *owl:someValuesFrom*, the constraint is lost.
- Classes that are a subclass of a cardinality constraint in an object property. The property is created with domain the class and range *owl:Thing*. 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:cardinality* constraint, the constraint is created as *owl:maxCardinality* instead of as *owl:cardinality*.
- Classes that are a subclass of a cardinality constraint in a datatype property. The property is created with domain the class and range *xsd:string*. The restriction is created with the *owl:allValuesFrom(xsd:string)* value constraint. In the case of the *owl:minCardinality* constraint, the constraint in the restriction is lost.
- Classes that are a subclass of a class intersection. The intersection is lost.

Class equivalences

- Classes equivalent to named classes. The *owl:equivalentClass* property is lost.
- Classes equivalent to a value constraint in an object property. The property is created with domain an anonymous class and range *owl:Thing*. The anonymous class is created as a subclass of the restriction and not as equivalent to the restriction. 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 equivalent to a cardinality constraint in an object property. The property is created with domain an anonymous class and range *owl:Thing*. The anonymous class is created as a subclass of the restriction and not as equivalent to the restriction. The restriction is created with the value constraint *owl:allValuesFrom(owl:Thing)*. In the case of the constraint *owl:minCardinality*, the constraint in the restriction is lost. In the case of the constraint *owl:cardinality*, it is created as *owl:maxCardinality* instead of as *owl:cardinality*.

- Classes equivalent to a cardinality constraint in a datatype property. The property is created with domain an anonymous class and range *xsd:string*. The anonymous class is created as a subclass of the restriction and not as equivalent to the restriction. The restriction is created with the value constraint *owl:allValuesFrom(xsd:string)*. In the case of the constraint *owl:minCardinality*, the constraint in the restriction is lost.
- Classes equivalent to a class intersection. The *owl:intersectionOf* and *owl:equivalentClass* properties are lost. An anonymous class is created.

Classes defined with set operators

- Classes that are intersection of other classes. The *owl:intersectionOf* property is lost.

Properties

- Object property hierarchies. The *rdfs:subPropertyOf* properties are lost. The properties are created with *owl:Thing* as domain and range.
- Datatype property hierarchies. The datatype properties are lost.
- Object properties without domain or range. When there are object properties without domain, the domain is created as *owl:Thing*. When there are object properties without range, the range is created as *owl:Thing* and the class is created as a subclass of the restriction in the property *owl:allValuesFrom(owl:Thing)*.
- Datatype properties without domain or range. When there are datatype properties without domain, the datatype property is lost. When there are properties without range, the range is created as *xsd:string* and the class is created as a subclass of the *owl:allValuesFrom(xsd:string)* restriction in the property.
- Object properties with domain and range. The class is created as a subclass of the *restriction(a:hasChild owl:allValuesFrom(a:Person))* restriction.
- Datatype properties with domain and range. The property is created with domain the class and range *xsd:string* and the class is created as a subclass of the restriction in the property *owl:allValuesFrom(xsd:string)*.
- Object properties with multiple domains. The object property is created with domain an anonymous class. The anonymous class is created as a subclass of all the domains and of the restriction *restriction(a:hasChild owl:allValuesFrom(a:Person))*.
- Object properties with multiple ranges. The object property is created with range an anonymous class. The anonymous class is created as a

subclass of all the ranges. The domain class is created as a subclass of the *restriction(a:hasChild owl:allValuesFrom(a:ANON))* restriction.

- Datatype properties with multiple domains. An anonymous class is created as a subclass of all the domains. The property is created with domain the class and range *xsd:string* and the class is created as a subclass of the restriction in the property *owl:allValuesFrom(xsd:string)*.

Relations between properties

- Equivalent object properties. The *owl:equivalentProperty* property is lost. The domain class is created as a subclass of *owl:allValuesFrom* restrictions in the properties.
- Equivalent datatype properties. The *owl:equivalentProperty* property is lost. The properties are created with range *xsd:string* and the domain class is created as a subclass of the *owl:allValuesFrom* restriction in the property.
- Inverse object properties. The *owl:inverseOf* property is lost. The domain class is created as a subclass of an *owl:allValuesFrom* restriction in the property. The property without domain and range is created with *owl:Thing* as domain and range.

Global cardinality constraints and logical property characteristics

- Transitive or symmetric object properties. The domain class is created as a subclass of an *owl:allValuesFrom* restriction in the property.
- Functional object properties. The class is created as a subclass of the restriction *restriction(a:hasHusband maxCardinality(1))*.
- Functional datatype properties. The properties are created with range *xsd:string* and the class is created as a subclass of the restriction in the property *owl:allValuesFrom(xsd:string)*. The class is created as a subclass of the restriction *restriction(a:hasName maxCardinality(1))*.
- Inverse functional object properties. The fact that the property is inverse functional is lost. The domain class is created as a subclass of an *owl:allValuesFrom* restriction in the property.

Individuals

- Individuals of a single class. The individuals remain the same.
- Individuals of multiple classes. The individual is created as instance of an anonymous class. The anonymous class is created as the subclass of the classes.

- Named individuals and object properties. The domain class is created as a subclass of an *owl:allValuesFrom* restriction in the property.
- Named individuals and datatype properties. The domain class is created as a subclass of an *owl:allValuesFrom* restriction in the property. The property is created with range *xsd:string*.
- Anonymous individuals and object properties. The anonymous individual is created as a named individual.
- Anonymous individuals and datatype properties. The anonymous individual is created as a named individual.

Individual identity

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

6.6.2. OWL interoperability results

Table 6.32 provides an overview of the updated interoperability between the tools; it shows the percentage of benchmarks in which the original and the resultant ontologies in an interchange are the same.

		DESTINATION							
		JE	PO	SP	KA	GA	ST	PF	WE
ORIGIN	JE	100	100	100	78	85	16	5	17
	PO	100	100	95	78	89	16	5	17
	SP	100	100	100	78	55	45	5	17
	KA	78	78	78	78	40	39		6
	GA	96	52	79	74	46	13	13	15
	ST	45	46	46	27	24	46		17
	PF	5	5			4	5	13	
	WE	17	18		6	16	17	12	17

Table 6.32: Updated percentage of identical interchanged ontologies.

In this table we can observe that in the 14 cases where the updated WebODE produces identical ontologies in *Step 1* (17% of the cases), such ontologies are correctly interchanged in the two directions with Jena, Protégé-OWL, SemTalk and WebODE itself. Besides, interoperability has slightly improved with the other tools.

Regarding robustness, the updated WebODE poses no execution problems when it processes the ontologies of the benchmark suite or when it processes ontologies generated by other tools. Furthermore, the execution problems that GATE encountered with some ontologies generated by WebODE have disappeared.

Table 6.33 provides a summary of the interoperability results of the updated WebODE with the other tools. Interoperability results between two tools, as presented in the previous section, have been grouped into categories, including the interchange from one tool to another and vice versa, and these results are restrictive. The results can be the following: *SAME* (*S*) when all the ontologies interchanged are the same, *DIFFERENT* (*D*) when at least one ontology interchanged is different and no execution errors exist, and *Non Executed* (-) when at least one ontology could not be interchanged because of an execution error.

The main effect of updating WebODE has been that the number of execution errors has diminished in WebODE and in the other tools. Moreover, WebODE can interchange correctly class hierarchies without cycles and instances of single classes with Jena, Protégé-OWL, SemTalk, and itself. It can also interchange class hierarchies without cycles with GATE, and instances of single classes with KAON2. These new interoperability results do not provide us with new issues besides those identified in the previous section.

To sum up, this section presents the improvement made on WebODE and the effect that this improvement has caused in its interoperability. Nevertheless, further improvement is needed to achieve higher interoperability both in WebODE and in the other tools.

Full interoperability of WebODE with the other tools is not possible because of the differences between its knowledge model and those of the other tools and the interchange language. But this does not prevent tools from maximizing the quantity of information that could be interchanged nor does it prevent them from posing execution errors.

Subgroups	WE-GA		WE-JE		WE-K2		WE-PF		WE-PO		WE-ST		WE-SP		WE-WE	
	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐	⇒	⇐
Class hierarchies																
Named class hierarchies without cycles	S	S	S	D	D	D	D	D	S	S	S	S	-	S	S	S
Named class hierarchies with cycles	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes subclass of a value constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes subclass of a cardinality constraint in an object property	D	D	D	D	-	-	-	-	D	D	D	D	-	D	D	D
Classes subclass of a cardinality constraint in a datatype property	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes subclass of a class intersection	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Class equivalences																
Equivalent named classes	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes equivalent to a value constraint in an object property	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes equivalent to a cardinality constraint in an object property	D	D	D	D	-	-	-	-	D	D	D	D	-	D	D	D
Classes equivalent to a cardinality constraint in a datatype property	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes equivalent to a class intersection	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Classes defined with set operators																
Classes intersection of other classes	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Property hierarchies																
Object property hierarchies	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Datatype property hierarchies	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Properties with domain and range																
Object properties without domain and range	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Object properties with domain and range	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Object properties with multiple domains or ranges	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Datatype properties without domain or range	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Datatype properties with domain and range	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Datatype properties with multiple domains	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Relations between properties																
Equivalent object and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Inverse object properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Global cardinality constraints and logical property characteristics																
Transitive object properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Symmetric object properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Functional object and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Inverse functional object properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Single individuals																
Instances	D	D	S	S	S	S	D	D	S	S	S	S	-	S	S	S
Instances of multiple classes	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Named individuals and properties																
Named individuals and object properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Named individuals and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Anonymous individuals and properties																
Anonymous individuals and object properties	D	D	D	D	-	-	-	-	D	D	D	D	-	D	D	D
Anonymous individuals and datatype properties	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Individual identity																
Equivalent individuals	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Different individuals	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D
Syntax and abbreviation																
Syntax and abbreviation	D	D	D	D	D	D	D	D	D	D	D	D	-	D	D	D

Table 6.33: Updated OWL interoperability results of WebODE.

Conclusions and future research lines

This chapter presents the conclusions of this work, focusing on the main advances made by the author to contribute to the state of the art of benchmarking Semantic Web technologies. As described in chapter 3, the objectives of the work here presented were the following:

- The development of a benchmarking methodology for Semantic Web technologies, grounded on existing benchmarking methodologies and practices in other areas, as general and open as possible so it can cover the broad range of Semantic Web technologies.
- The application of this methodology to the case of benchmarking the interoperability of Semantic Web technologies using RDF(S) and OWL as interchange languages; such application involved the provision of specific methods, technology, and benchmark suites that support these benchmarkings.

The next sections present the conclusions related to the main contributions made to the state of the art by this thesis.

7.1. Development and use of the benchmarking methodology

Benchmarking of Semantic Web technologies is an arduous task because there is no software benchmarking methodology and current evaluation and improvement methodologies are difficult to use with Semantic Web technologies because they are not defined in detail.

One of the contributions of this thesis is the development of a benchmarking methodology for Semantic Web technologies. This methodology is collaborative and open, and has been defined having as a starting point well-known evaluation and benchmarking methodologies from other areas and reusing the common tasks of these methodologies.

This methodology has not been formally evaluated. Nevertheless, as we show below, it has been validated by checking that it meets each and every necessary and sufficient condition that every methodology should satisfy and its applicability has been proved by using it in the benchmarking activities performed in the Knowledge Web Network of Excellence. We cannot ensure that the methodology is valid in every use scenario, but further validation of the methodology will be possible in future benchmarking activities in different settings.

Furthermore, as presented at the end of this section, using the benchmarking methodology has provided us with ideas on how to improve the methodology and with recommendations for benchmarking.

The benchmarking methodology meets the **necessary conditions** that every methodology should satisfy, which were stated in section 4.1. Specifically, the benchmarking methodology is:

- Complete because it allows considering any type of Semantic Web technology and any possible criteria to evaluate this kind of technology.
- Efficient because it is described in a simple way, and it is easy to understand and use it with no special effort. Furthermore, being general as it is, it can be adapted to the personal and working conditions of different actors.
- Effective because following the different tasks of the benchmarking methodology and producing the expected results in each of these tasks ensures that progress is being achieved and that the goals of the benchmarking are met at the end of it.
- Consistent because different people performing the same (or similar) benchmarking activities will propose different solutions, but they should not obtain contradictory conclusions if the experiments are defined correctly.
- Responsive because though the techniques of the different benchmarking tasks have not been defined on purpose, answers are given to the rest of the questions for every task, not placing too many restrictions to make the methodology flexible.
- Advertent because the benchmarking problem is divided into different tasks, each of them producing some partial results, which allows the evaluation of the benchmarking at the end of each task and of the whole benchmarking.
- Finite because, although benchmarking is a continuous process, each benchmarking iteration takes place in a finite period of time.
- Discernible because the number of components of the benchmarking methodology is small, and they allow different representations and visualizations.

- A technological methodology, as it is related to the development and assessment of software.
- Transparent because the actors, inputs, and outputs of each task are specified. Therefore, we can know, in every moment and according to the outputs already produced, the current benchmarking task being performed and who is performing what in this task.

The benchmarking methodology also meets the **sufficient conditions** of a methodology for benchmarking Semantic Web technologies, which were stated in section 4.1. Specifically, the benchmarking methodology is:

- Benchmarking-oriented because its tasks allow obtaining a continuous improvement of the tools and getting the best practices performed when developing these tools.
- Grounded on existing practices because it has been defined by combining tasks of existing methodologies.
- Collaborative because it contemplates the participation and consensus of different actors from different organizations.
- Open because it does not limit the types of Semantic Web technologies or the entities (software products, services or processes) to be considered in benchmarking, nor the phase of the software life cycle when benchmarking is performed, nor does it limit who is responsible for carrying out the benchmarking.
- Usable because it is clearly documented and using it does not involve a high effort.

The **applicability of the benchmarking methodology** has been proved in the Knowledge Web benchmarking activities where this methodology has been used for benchmarking ontology development tools (as seen in section 4.4 and in chapters 5 and 6), ontology alignment tools [Euzenat *et al.*, 2004b], and reasoners [Huang *et al.*, 2007]. An approach to benchmarking the efficiency and scalability of ontology development tools using this methodology can be found in [García-Castro and Gómez-Pérez, 2005a].

We have also proven that it is feasible to evaluate and improve different Semantic Web technologies using a common method and following a problem-focused approach instead of a tool-focused approach. This is the case of the two case studies shown in this thesis, while the other benchmarking activities over ontology alignment tools, reasoners and ontology development tools were tool-focused.

These practical uses of the methodology have provided us with the following lessons and **ideas on how to improve the methodology**:

- Some task names, although descriptive, can cause confusion if interpreted in different contexts. For example, in the *Subject and metrics identification* task, the word “subject” is too specific for experimentation. Also, the *Participant identification* task is sometimes confused with the *Partner selection* task.
- In the methodology, benchmarking is initiated by an organization, but it may occur that it is initiated by a group of organizations, as it happens with benchmarking activities organized by research communities.
- When benchmarking is open to any organization, the benchmarking proposal can be a web page, which maximizes its dissemination. If instead of a static web page, the proposal is included in a wiki, then the proposal can be edited by the participants collaboratively.
- The benchmarking proposal should not include detailed information specific to one organization and not relevant to the other organizations, namely, information on the members of the organization involved in the benchmarking, or information on detailed benefits and costs of any organization. Therefore, we suggest to have two documents instead of one: one with the benchmarking proposal and the other with organization-specific information.
- Information about the benchmarking planning should be included in the proposal and agreed on in the *Partner selection task*. This is so because quite often the decision of an organization to participate in the benchmarking depends on the benchmarking planning.

Furthermore, after using the methodology in these two case studies, we can provide the following **recommendations for benchmarking**:

- First of all, the participation of relevant experts of the community during the whole benchmarking process is crucial, and the inclusion of the best-in-class tools is a must, even in those cases in which the organizations that develop these tools do not participate in the process.
- Benchmarking takes long time as it requires tasks that are not immediate: announcements, agreements, etc. Therefore, its planning should consider a realistic duration of the benchmarking and should provide the necessary resources.
- The effort to be devoted to benchmarking is a main criteria for any organization (especially companies) when it has to decide whether to participate in it. Resources are needed mainly in four tasks: benchmarking organization, definition of the experiment, execution of the experiments, and analysis of the results. Therefore, the tasks to be performed in the benchmarking, particularly the experiment-related tasks, should be automated as much as possible.

7.2. Benchmarking interoperability

Current evaluation and benchmarking activities over Semantic Web technologies are scarce and a hindrance to the full development and maturity of these technologies. The Semantic Web needs to produce methods and tools for evaluating its technologies at large scale and in an easy and economical way, which requires defining technology evaluations focused on their reusability.

Before starting this thesis, there were no methods, benchmark suites and tools that could be reused for evaluating and benchmarking the interoperability of Semantic Web technologies using an interchange language.

This thesis contributes to the state of the art by defining the UPM Framework for Benchmarking Interoperability (UPM-FBI). This framework considers two approaches, one manual and the other automatic, for benchmarking interoperability using an interchange language and provides tools and benchmark suites to support both approaches.

As the UPM-FBI is publicly available, it can be used both by tool developers to evaluate and improve the interoperability of their tools, providing these tools have import and export functionalities, and by ontology engineers to select the appropriate tool for their ontology development activities.

Next, we present some conclusions regarding the resources provided by the UPM-FBI, i.e., the benchmark suites, the manual and automatic approaches and the tools that support it.

The **four benchmark suites** defined in this thesis satisfy the eight desirable properties identified for benchmark suites, mentioned in section 2.4:

- *Accessibility.* The complete definition of the benchmark suites as well as all the information relevant to the benchmarking activities is accessible to anyone in public web pages. These pages also include the results obtained when using the benchmark suites. Thus, anyone can use them and compare their results with the ones available.
- *Affordability.* The costs of using the benchmark suites and analysing their results are mainly in terms of human resources. In order to reduce these costs and facilitate the work, a clear definition of the benchmark suites was provided as well as data collection and analysis mechanisms, such as templates to fill in the results in the RDF(S) Interoperability Benchmarking or the IBSE tool that executes the experiments in the OWL Interoperability Benchmarking.
- *Simplicity.* The benchmark suites are simple and interpretable because different ways of defining each benchmark are provided, e.g., in natural language, graphically, etc. These benchmark suites and their results are also clearly documented, having a common structure and use.
- *Representativity.* Although the different benchmarks that compose the benchmark suites are not exhaustive or represent real-world ontologies,

they embody the different ontology structures commonly used when developing these ontologies; a first evaluation using simple ontologies is needed before an evaluation using more complex ones.

- *Portability.* The benchmark suites are defined at a high level of abstraction, so they are not biased towards a certain tool or tools. Therefore, they can be executed on a wide variety of environments.
- *Scalability.* The benchmark suites are scalable and can work with tools at different levels of maturity. Also, as their benchmarks are grouped according to the different ontology components that they manage, it is quite easy to increase or decrease the number of benchmarks by taking into account new components or only certain components of interest, respectively.
- *Robustness.* Since the results of the benchmark suites only depend on the algorithms implemented to perform the import and export of ontologies, they are not influenced by factors irrelevant to the study. Furthermore, running the benchmark suites with the same version of the tools will always produce the same results.
- *Consensus.* The benchmark suites have been assessed and agreed on by the members of Knowledge Web and by the benchmarking partners, who are experts in the domain of ontology translation and interoperability.

It must be noted that the benchmark suites here presented have been defined with the goal of evaluating interoperability. Therefore, even if these benchmark suites can be used to evaluate tool importers and exporters, they are not exhaustive for these tasks and should be extended. An exhaustive evaluation of the RDF(S)/OWL import capabilities of a tool should take into account the whole RDF(S)/OWL model, whereas an exhaustive evaluation of the export capabilities of a tool should take into account the whole knowledge model of the tool.

Since in each benchmarking we followed one of the **two experiment approaches** included in the UPM-FBI (i.e., the manual approach in the RDF(S) Interoperability Benchmarking and the automatic approach in the OWL Interoperability Benchmarking), we can now make a comparative analysis of these two approaches:

- Carrying out experiments manually and analysing the results involves spending significant resources. Therefore, experiments in benchmarking should be automated as much as possible. However, full automation of the result analysis is not possible since it requires a person to interpret them; nevertheless, the automatic generation of different visualizations and summaries of the results in different formats (textual, tabular, or graphical) allows drawing some conclusions rapidly.
- Some research institutions and companies chose not to participate in the benchmarking activities because they could not afford the expenses. Thus,

their absence affected the number of participants in the RDF(S) Interoperability Benchmarking, which was fairly low (3 organizations). However, the number of participants was higher (7 organizations) in the OWL Interoperability Benchmarking, because automating the experiment and facilitating the inclusion of new tools in the experiments, besides lowering the benchmarking costs, favoured the participation of new organizations.

- The results in the manual approach depend on the expertise of the people performing the experiments and can be influenced by human errors. But, on the other hand, the automatization of the experiment can minimise human errors and, whenever human intervention is needed, mechanisms should be set up to detect this kind of errors.
- The automatic approach is more flexible and extensible than the manual approach. Cyclic interoperability experiments (from one origin tool to a destination one and then back to the origin tool) or chained interoperability experiments (from one origin tool to an intermediate one and then to a third tool) are very easy to perform following the automatic approach, but they are too expensive to carry out in the manual approach.
- Current ontology comparers have problems when comparing ontologies, and they only compare ontologies from the *Lexical* to the *Conceptual* levels, according to the levels presented in section 1.3.3. Therefore, to find differences between two ontologies at the pragmatic level we need an expert.
- In the manual approach, the resultant ontology is compared with the expected one, defined according to the developer specifications. The automatic approach, on the other hand, compares the resultant ontology with the original ontology and, therefore, it does not regard the tool specifications. This behaviour causes that, in the automatic approach, tools are sometimes said to have errors when they function as it was expected by their developers.
- Manual experiments are better suited to obtain the practices that lead to the results than automatic experiments. In the manual approach, these practices were obtained both by providing specific questions to the experimenters in order to identify the practices used to develop the tools and by allowing the experimenters to comment on the tools behaviour. This was not possible with the automatic approach, since after the experiment we only obtained raw results from the tools.
- Manual experiments are best suited to locate problems in the tools, since they provide separate results for the import and export functionalities, whereas automatic experiments provide results for the joint import and export operation, without detecting whether the problem occurs when importing or exporting the ontology.

To sum up, the manual approach has some advantages over the automatic approach and vice versa. The effort required to carry out the experiments and the quality of the results depend on the human involvement and while automatic experimentation is cheaper, more flexible and extensible, the quality of human-generated results is higher.

Therefore, the approach to follow will depend on the specific needs of the benchmarking; however, the automatic approach is recommended because it is preferable a lesser cost in the benchmarking and a higher number of participants to an increase in the result quality provided by the manual approach.

Furthermore, the definition of the experiments in these two approaches is independent of the interchange language; the election of a certain language only affects the ontologies to be used as input for the experiments. Therefore, the manual approach can also be used to evaluate interoperability using OWL as interchange language and the automatic approach can also be used to evaluate interoperability using RDF(S) as interchange language.

We have also developed **the IBSE tool**, an easy-to-use tool for large-scale evaluations of the interoperability of Semantic Web technologies when using an interchange language. This tool can be used in other scenarios, using any group of ontologies as input or other languages as interchange. Right now the tool allows performing experiments using RDF(S) as the interchange language and *rdf-utils*¹ as ontology comparer; other tools, however, should have to implement the corresponding interface in the IBSE tool and then use the RDF(S) Import Benchmark Suite, described in section 5.1.1, as ontology dataset.

The IBSE tool can be used to evaluate tools, either in the early stages of their development or when the development has finished, and to monitor their improvement. It 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 importers and exporters.

7.3. RDF(S) and OWL interoperability results

At the moment of starting this thesis, the limits of the interoperability between the different Semantic Web tools were unknown.

The main contribution to solve this problem is the assessment of the current interoperability of several well-known Semantic Web tools, six in the RDF(S) Interoperability Benchmarking and nine in the OWL one. Such assessments have provided us with results about the detailed behaviour of the tools not just when they interoperate with other tools but also when they import and export RDF(S) and OWL ontologies.

The benchmarking results are publicly available in the Web, so anyone can use them. Nevertheless, it must be noted that these results are valid for the specific versions of the tools in which the experiments were performed and, because the development of these tools continues, the results are expected to change. This highlights the need of a continuous evaluation of the technology.

¹version 0.3b <http://wymiyg.org/rdf-utils/>

The main conclusion drawn from the interoperability results is that in general **interoperability between the tools is very low**, even in interchanges between a tool and the tool itself, and the clusters of interoperable tools are minimal.

From the RDF(S) interoperability results in section 5.5, we see that only Corese, Jena, KAON, Sesame, and WebODE can interoperate with themselves using RDF(S) as the interchange language and that the only clusters of RDF(S)-interoperable tools are Corese with Jena and with Sesame, and Protégé-Frames with WebODE; using in all cases the common knowledge model components that both tools can model.

From the OWL interoperability results in section 6.5, we see that only Jena, Protégé-OWL, and SWI-Prolog can interoperate with themselves using OWL as the interchange language and the only clusters of OWL interoperable tools are Jena with Protégé-OWL and Jena with SWI-Prolog.

Furthermore, **interoperability using an interchange language highly depends on the knowledge models of the tools**. This said, we can add that interoperability is better when the knowledge model of the tools is similar to that of the interchange language. This can be seen in the results where the tools that better interoperate are those whose knowledge models fully cover the knowledge model of the interchange language.

In the cases where the knowledge models differ, interoperability can be only achieved by means of lightweight ontologies. For example, when Protégé-Frames interoperates with WebODE using RDF(S) as the interchange language, they can only interchange ontologies that include a limited set of components, and they are not able to use a richer expressivity as their knowledge models allow.

This panoramic, although disappointing, may serve to promote the second of our goals: the improvement of the tools. Even though tool improvement is out of our scope right now because each tool is developed by independent organizations, we here show how we have achieved a great improvement in WebODE and hope, nevertheless, that the results provided may help improve the other tools.

We can add that some of the participating tools have been improved even before the *Improvement* phase of the methodology. And because the goal was improvement, modifications on the participating tools were allowed at any time and, in some cases, tools were improved while the experiments were being executed.

Therefore, real interoperability in the Semantic Web requires the involvement of tool developers. The developers of the tools participating in the benchmarking activities have been informed of the results of these activities and of the recommendations proposed for improving their tools.

After analysing the results, we checked that **the interoperability problem not only depends on the ontology translation problem but also on some robustness and specification problems** (i.e., development decisions taken by their developers).

If we consider the levels of ontology translation problems described in section 1.3.3, the results show differences in all levels (*Lexical*, *Syntactic*, *Paradigm*,

Terminological, Conceptual, and Pragmatic). Nevertheless, we can observe that most of the differences found in the interoperability results occur in the *Conceptual* level.

Furthermore, it cannot be said that there is a group of “typical” interoperability problems, since the interoperability results highly depend on the tools that participate in the interchange and, on the other hand, the behaviours of each tool are quite different.

Ontology developers can influence the results of interchanging ontologies between different tools. Therefore, the following **recommendations for ontology developers** have been extracted from the analysis of the benchmarking results:

- Ontology developers should be aware of the components that can be represented in the knowledge models of the tools and in the interchange languages. Hence, they should try to use the common components of these tools in their ontologies to avoid the already-known knowledge losses.
- Ontology developers should also be aware of the equivalences and differences between the knowledge models of the tools and the knowledge model of the interchange language. For example, in Protégé multiple domains in template slots are considered the union of all the domains, while in RDF(S) multiple domains in properties are considered the intersection of all the domains; in WebODE instance attributes are local to a single concept, while in RDF(S) properties are global and can be used in any class.
- It is not recommended to name resources by including in their names spaces or any character that is restricted in the RDF(S), OWL, URI or XML specifications.

Interoperability between ontology development tools using an interchange language depends on how the importers and exporters of these tools work. In their turn, how these importers and exporters work depends on the development decisions made by tool developers, who are different people with different needs. Therefore, it is not straightforward to provide general recommendations for developers since many issues are involved. However, some **recommendations for Semantic Web software developers** can be extracted from the analysis of the benchmarking results:

- The first requirement for achieving interoperability is that the importers and exporters of the tools be robust and work correctly when dealing with unexpected inputs. Although this is an evident recommendation, the results show that this requirement is not always fulfilled by the tools and that some tools even crash when they import some combinations of components.
- Above all, tools should work correctly with the combinations of components that can be present in the interchange language but that cannot be

modelled in them; for example, cycles in class and property hierarchies cannot be modelled in ontology development tools. Nevertheless, these tools should be able to import these hierarchies by eliminating the cycles.

- When exporting components commonly used by ontology development tools, they should be completely defined in the file; for example, in RDF(S), metaclasses and classes in class hierarchies should be defined as instances of *rdfs:Class*, properties should be defined as instances of *rdf:Property*, etc.
- Exporting complete definitions of components rarely used by the tools can cause problems if these are later imported by other tools; for example, not every tool deals with datatypes defined as instances of *rdfs:Datatype* in the file or with *rdf:datatype* attributes in properties.
- Every exported resource should have a namespace if the document does not define a default namespace.
- In a few cases, a development decision will improve interoperability with some tools but produce loss with others; for example, when exporting to RDF(S) classes that are instances of a metaclass, some tools require that the class be defined as instance of *rdfs:Class* while some other tools require the opposite, being the two options correct.
- The collateral consequences of the development decisions should be analysed by the tool developers. For example, if a datatype is imported as a class in the ontology, then the literal values of this datatype should be imported as instances in the ontology, which would complicate the management of these values.
- Tool developers and ontology developers should be aware of the semantic equivalences and differences between the knowledge models of their tool and the interchange language; on the other hand, tools should notify the user when the semantics is changed.

7.4. Open research problems

Even though this thesis presents significant contributions to the field of benchmarking Semantic Web technologies, there are still some open research problems that have not been contemplated in this thesis, or that have come out as a consequence of the advances proposed in it. These open research problems, which mainly correspond to the restrictions already identified in chapter 3, are described below.

- The benchmarking methodology for Semantic Web technologies proposed in this thesis has only been used with such technologies. As the methodology is general enough to handle any type of software, it can be straightforward to use it for benchmarking software from outside the Semantic Web

area and to analyse the appropriateness of the methodology for software in general.

- Another possible line of research would be to analyse whether the methodology can also be applied to software processes or services and not just to software products. To this end, new benchmarking activities should be conducted over this type of software, analysing the changes needed in the current methodology.
- In order to obtain a general methodology, we disregarded the specific techniques or tools needed for carrying out each task of the methodology. One way of completing and improving the benchmarking methodology would be to identify the different techniques to be used in each of the tasks of the process and to provide software to support these tasks.
- In the benchmarking activities described in this thesis, only few tools participated if we take into account the number of tools existing in the Semantic Web. We think that it would be desirable for the future to continue these benchmarking activities with a higher number of tools.
- In future iterations of the benchmarking activities, the evaluation should be updated to be more exhaustive, taking into account the full knowledge models of the tools or the interchange languages, or including real-world ontologies in the benchmark suites.
- The experiments could be also extended by incorporating other ontology interchanges from one origin tool to a destination one, as for example cyclic interoperability experiments (from one origin tool to a destination one and then back to the origin tool) or chained interoperability experiments (from one origin tool to an intermediate one and then to a third one).
- The main topic of this thesis is the problem of interoperability between Semantic Web technologies using an interchange language. Other benchmarking activities should be conducted over these technologies, considering other approaches to the interoperability problem, exploiting other evaluation criteria (i.e., efficiency, scalability, robustness, etc.), or centering the benchmarking on the needs of the users of Semantic Web technologies.
- To increase the usability of the benchmarking results, the main improvement would be to facilitate effective ways of analysing and exploiting the results by means of a web application, so that users could perform complex analyses of these results. In the case of the RDF(S) Interoperability Benchmarking results, this requires a previous translation of the results from the spreadsheets where they were collected to a machine-processable format. The IRIBA² application, which is currently under development,

²<http://knowledgeweb.semanticweb.org/iriba/>

allows analysing the RDF(S) interoperability results of the tools in different moments. Regarding the OWL interoperability results, the IBSE tool already produces HTML pages that summarize these results. Nevertheless, it would be very useful to also have a web application that allows a dynamic and personalised analysis of the OWL interoperability results.

- It would also be quite convenient that the IBSE tool provided results that were easier to analyse and that included specific visualizations of results for tools whose internal knowledge model does not correspond with the interchange language. With such tools, the analysis of the results is not straightforward and sometimes triples are inserted or removed, as intended by their developers, but this correct functioning is difficult to evaluate or to distinguish in the current results.
- Further technical improvements of the IBSE tool include both changing the OWL comparer and integrating IBSE with testing infrastructures, such as JUnit³.

7.5. Dissemination of results

To conclude this thesis, it is important to remark that parts of this thesis have been internationally disseminated.

A first version of the **benchmarking methodology for Semantic Web technologies** was presented at the *4th International Semantic Web Conference* (ISWC2005) [García-Castro and Gómez-Pérez, 2005a]. The paper also included an approach for benchmarking the efficiency and scalability of ontology development tools. Although this thesis does not deal with such approach, further details of it were presented at the *3rd International Workshop on Evaluation of Ontology-based Tools* (EON2004) [García-Castro and Gómez-Pérez, 2004] and as posters at the *4th International Conference on Language Resources and Evaluation* (LREC2004) [Corcho *et al.*, 2004] and at the *1st International Workshop on Practical and Scalable Semantic Systems* (PSSS2003) [Corcho *et al.*, 2003a].

The **RDF(S) Interoperability Benchmarking** was first presented at the *1st International Workshop on Scalable Semantic Web Knowledge Based Systems* (SSWS2005) [García-Castro and Gómez-Pérez, 2005b] along with the manual approach of the UPM-FBI and the method followed for defining the RDF(S) Import Benchmark Suite.

The instantiation of the benchmarking methodology for the RDF(S) Interoperability Benchmarking and the definition of the three benchmark suites used in it were presented at the *3rd European Semantic Web Conference* (ESWC2006) [García-Castro and Gómez-Pérez, 2006a].

A whole summary of the RDF(S) Interoperability Benchmarking, including how the methodology was instantiated and the interoperability results of the

³<http://www.junit.org/>

tools, was presented at the *19th International Conference on Software Engineering and Knowledge Engineering* (SEKE2007) [García-Castro *et al.*, 2007c]. The results in the RDF(S) Interoperability Benchmarking for the Protégé tool were also presented at the *9th International Protégé Conference* (Protégé2006) [García-Castro and Gómez-Pérez, 2006b].

The **OWL Interoperability Benchmarking** and the method followed to define the OWL Import Benchmark Suite were first presented at the *2nd International Workshop OWL: Experiences and Directions 2006* (OWL2006) [David *et al.*, 2006]. At the 3rd edition of this same workshop (OWL2007) [García-Castro *et al.*, 2007b] the automatic approach of the UPM-FBI and the IBSE tool were presented.

A whole summary of the OWL Interoperability Benchmarking, including how the methodology was instantiated and the interoperability results of the tools, was presented at the *2nd IEEE International Conference on Semantic Computing* (ICSC2008) [García-Castro and Gómez-Pérez, 2008a].

An overview of the content of this thesis has been presented at two **Ph.D. symposia**, namely, the *1st Knowledge Web PhD Symposium* (KWEPSY2006) [García-Castro, 2006a] and the *Doctoral Consortium of the XII Conferencia de la Asociación Española para la Inteligencia Artificial* (CAEPIA-TTIA 2007) [García-Castro, 2007a].

The candidate has also been invited to give two **keynote speeches** related to the work presented in this thesis: at the *2nd International Workshop on Scalable Semantic Web Knowledge Base Systems* (SSWS2006) [García-Castro, 2006c], the author presented a summary of the state of the art of evaluation and benchmarking of Semantic Web technologies, the benchmarking methodology for Semantic Web technologies, and an example of how to use it; at the *IV Seminario Internacional Tecnologías Internet* (SITI2006) [García-Castro, 2006b], he presented the problem of interoperability between Semantic Web technologies and the RDF(S) and OWL Interoperability Benchmarkings.

A **book chapter** describing the state of the art of evaluation and benchmarking of Semantic Web technologies, including the benchmarking methodology for Semantic Web technologies and an example of how to use it, are included in the *Semantic Web Engineering in the Knowledge Society* book [García-Castro and Gómez-Pérez, 2008b], which will be published in 2008.

Finally, further details of the work presented in this thesis are included in different **deliverables** within the Knowledge Web [García-Castro *et al.*, 2004, García-Castro, 2005, García-Castro *et al.*, 2006, García-Castro *et al.*, 2007a] and in the NeOn [García-Castro, 2007b] projects.

Bibliography

- [Ahmed and Rafiq, 1998] P.K. Ahmed and M. Rafiq. Integrated benchmarking: a holistic examination of select techniques for benchmarking analysis. *Benchmarking for Quality Management and Technology*, 5(3):225–242, 1998.
- [Andersen and Pettersen, 1996] B. Andersen and P.G. Pettersen. *The Benchmarking Handbook: Step by step instructions*. Chapman & Hall, London, 1996.
- [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.
- [Barrasa, 2007] J. Barrasa. *Modelo para la definición automática de correspondencias semánticas entre ontologías y modelos relacionales*. PhD thesis, Universidad Politécnica de Madrid. Facultad de Informática, January 2007.
- [Basili *et al.*, 1986] V.R. Basili, R.W. Selby, and D.H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering* 12, (7):733–743, January 1986.
- [Basili, 1985] V.R. Basili. Quantitative evaluation of software methodology. In *1st Pan-Pacific Computer Conference*, Melbourne, Australia, September 1985.
- [Berners-Lee *et al.*, 2001] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
- [Boehm *et al.*, 1976] B.W. Boehm, J.R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering (CSE1976)*, pages 592–605, San Francisco, California, United States, 1976. IEEE Computer Society Press.
- [Boehm *et al.*, 2005] B. Boehm, H.D. Rombach, and M.V. Zelkowitz. *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

- [Bontcheva *et al.*, 2004] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to meet new challenges in language engineering. *Natural Language Engineering*, 10(3-4):349–373, 2004.
- [Bouquet *et al.*, 2004] P. Bouquet, M. Ehrig, J. Euzenat, E. Franconi, P. Hitzler, M. Krötzsch, L. Serafini, G. Stamou, Y. Sure, and S. Tessaris. D2.2.1 Specification of a common framework for characterizing alignment. Technical report, Knowledge Web, December 2004.
- [Brachmann and Levesque, 1985] R. Brachmann and H. Levesque. *Readings in Knowledge Representation*, chapter A Fundamental Tradeoff in Knowledge Representation and Reasoning, pages 31–40. Morgan Kaufmann, San Mateo, 1985.
- [Brickley and Guha, 2004] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004, 2004.
- [Broekstra *et al.*, 2002] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*, volume 2342, pages 54–68. Springer, 2002.
- [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.
- [Burnstein, 2003] I. Burnstein. *Practical Software Testing: A Process-Oriented Approach*. Springer Verlag, 2003.
- [Calvo and Gennari, 2003] F. Calvo and J.H. Gennari. Interoperability of Protégé 2.0 beta and OilEd 3.5 in the domain knowledge of osteoporosis. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003.
- [Camp, 1989] R. Camp. *Benchmarking: The Search for Industry Best Practice that Lead to Superior Performance*. ASQC Quality Press, Milwaukee, 1989.
- [Carroll and Roo, 2004] J.J. Carroll and J. De Roo. OWL Web Ontology Language Test Cases. Technical report, W3C, February 2004.
- [Cavano and McCall, 1978] J.P. Cavano and J.A. McCall. A framework for the measurement of software quality. In *Proceedings of the software quality assurance workshop on functional and performance issues*, pages 133–139, 1978.
- [Corby and Faron-Zucker, 2002] O. Corby and C. Faron-Zucker. Corese: A Corporate Semantic Web Engine. In *Proceedings of the International Workshop on Real World RDF and Semantic Web Applications, 11th International World Wide Web Conference*, Hawai, USA, May 2002.

- [Corcho *et al.*, 2003a] O. Corcho, R. García-Castro, and A. Gómez-Pérez. Towards a benchmark of the ODE API methods for accessing ontologies in the WebODE platform. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS2003) located at the 2nd International Semantic Web Conference (ISWC2003)*, Florida, USA, October 20th 2003.
- [Corcho *et al.*, 2003b] O. Corcho, A. Gómez-Pérez, D.J. Guerrero-Rodríguez, D. Pérez-Rey, A. Ruiz-Cristina, T. Sastre-Toral, and M.C. Suárez-Figueroa. Evaluation experiment of ontology tools' interoperability with the WebODE ontology engineering workbench. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003.
- [Corcho *et al.*, 2004] O. Corcho, R. García-Castro, and A. Gómez-Pérez. Benchmarking ontology tools. a case study for the WebODE platform. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC2004)*, Lisbon, Portugal, May 26th 2004.
- [Corcho, 2005] O. Corcho. *A layered declarative approach to ontology translation with knowledge preservation*, volume 116 of *Frontiers in Artificial Intelligence and its Applications*. IOS Press, January 2005.
- [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.
- [Davies *et al.*, 2006] J. Davies, R. Studer, and P. Warren, editors. *Semantic Web Technologies - trends and research in ontology-based systems*. John Wiley & Sons, JUN 2006.
- [Dou *et al.*, 2004] D. Dou, D. McDermott, and P. Qi. Ontology translation on the Semantic Web. *Journal of Data Semantics*, 2(3360):35–57, 2004.
- [Duval, 2004] E. Duval. Learning technology standardization: Making sense of it all. *International Journal on Computer Science and Information Systems*, 1(1):33–43, 2004.
- [Erdmann and Wenke, 2007] M. Erdmann and D. Wenke. D6.6.1 Realisation and early evaluation of basic NeOn tools in NeOn toolkit v1. Technical report, NeOn project, August 2007.
- [Euzenat *et al.*, 2004a] J. Euzenat, T. Le Bach, J. Barrasa, P. Bouquet, J. De Bo, R. Dieng-Kuntz, M. Ehrig, M. Hauswirth, M. Jarrar, R. Lara, D. Maynard, A. Napoli, G. Stamou, H. Stuckenschmidt, P. Shvaiko, S. Tessaris, S. Van Acker, and Ilya Zaihrayeu. D2.2.3 State of the art on ontology alignment. Technical report, Knowledge Web, June 2004.

- [Euzenat *et al.*, 2004b] J. Euzenat, R. García-Castro, and M. Ehrig. D2.2.2 Specification of a benchmarking methodology for alignment techniques. Technical report, Knowledge Web, December 2004.
- [Fernandez *et al.*, 2001] P. Fernandez, I.P. McCarthy, and T. Rakotobe-Joel. An evolutionary approach to benchmarking. *Benchmarking: An International Journal*, 8(4):281–305, 2001.
- [Fillies and Weichhardt, 2005] C. Fillies and F. Weichhardt. Semantically correct Visio drawings. In *Proceedings of the Workshop on User Aspects of the Semantic Web (UserSWeb2005)*, pages 85–92, May 2005.
- [Fillies, 2003] C. Fillies. SemTalk EON2003 Semantic Web Export / Import Interface Test. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003.
- [García-Castro and Gómez-Pérez, 2004] R. García-Castro and A. Gómez-Pérez. A benchmark suite for evaluating the performance of the WebODE ontology engineering platform. In *Proceedings of the 3rd International Workshop on Evaluation of Ontology-based Tools (EON2004) located at the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 8th 2004.
- [García-Castro and Gómez-Pérez, 2005a] R. García-Castro and A. Gómez-Pérez. Guidelines for Benchmarking the Performance of Ontology Management APIs. In Y. Gil, E. Motta, R. Benjamins, and M. Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, number 3729 in LNCS, pages 277–292, Galway, Ireland, November 2005. Springer-Verlag.
- [García-Castro and Gómez-Pérez, 2005b] R. García-Castro and A. Gómez-Pérez. A method for performing an exhaustive evaluation of RDF(S) importers. In *Proceedings of the Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2005)*, number 3807 in LNCS, pages 199–206, New York, USA, November 2005. Springer-Verlag.
- [García-Castro and Gómez-Pérez, 2006a] R. García-Castro and A. Gómez-Pérez. Benchmark suites for improving the RDF(S) importers and exporters of ontology development tools. In Y. Sure and J. Domingue, editors, *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, number 4011 in LNCS, pages 155–169, Budva, Montenegro, June 2006. Springer-Verlag.
- [García-Castro and Gómez-Pérez, 2006b] R. García-Castro and A. Gómez-Pérez. Interoperability of Protégé using RDF(S) as interchange language. In *Proceedings of the 9th International Protégé Conference (Protégé2006)*, Stanford, USA, July 2006.

- [García-Castro and Gómez-Pérez, 2008a] R. García-Castro and A. Gómez-Pérez. Large-scale benchmarking of the OWL interoperability of Semantic Web technologies. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC2008)*, Santa Clara, CA, USA, August 4-7 2008. IEEE Computer Society.
- [García-Castro and Gómez-Pérez, 2008b] R. García-Castro and A. Gómez-Pérez. *Semantic Web Engineering in the Knowledge Society*, chapter Benchmarking in the Semantic Web. Idea Group, To appear in 2008.
- [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 *et al.*, 2007a] R. García-Castro, S. David, and J. Prieto-González. D1.2.2.1.2 Benchmarking the interoperability of ontology development tools using OWL as interchange language. Technical report, Knowledge Web, September 2007.
- [García-Castro *et al.*, 2007b] R. García-Castro, A. Gómez-Pérez, and J. Prieto-González. IBSE: An OWL Interoperability Evaluation Infrastructure. In *Proceedings of the OWL: Experiences and Directions 2007 workshop (OWL2007)*, Innsbruck, Austria, June 6-7 2007.
- [García-Castro *et al.*, 2007c] R. García-Castro, A. Gómez-Pérez, and Y. Sure. Benchmarking the RDF(S) interoperability of ontology tools. In *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE2007)*, pages 410–415, Boston, Massachusetts, USA, July 9-11 2007. Knowledge Systems Institute Graduate School.
- [García-Castro *et al.*, 2007d] R. García-Castro, O. Muñoz-García, M.C. Suárez-Figueroa, A. Gómez-Pérez, S. Costache, D. Maynard, S. Dasiopoulou, R. Palma, V. Novacek, F. Lécué, Y. Ding, M. Kaczmarek, R. Piskac, D. Zyskowski, J. Euzenat, M. Dzbor, L. Nixon, A. Léger, T. Vitvar, M. Zaremba, and J. Hartmann. D1.2.5 Architecture of the Semantic Web Framework v2. Technical report, Knowledge Web, December 2007.
- [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, 2006a] R. García-Castro. Benchmarking Semantic Web technology. In *Proceedings of the 1st Knowledge Web PhD Symposium (KWEPSY2006)*, Budva, Montenegro, June 2006.
- [García-Castro, 2006b] R. García-Castro. Keynote: Tecnologías de la Web Semántica: cómo funcionan y cómo interoperan. In *Proceedings of the 4th Seminario Internacional Tecnologías Internet (SITI2006)*, Popayán, Colombia, 5-7 October 2006 2006.
- [García-Castro, 2006c] R. García-Castro. Keynote: Towards the improvement of the Semantic Web technology. In *Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006)*, Athens GA, USA, November 2006.
- [García-Castro, 2007a] R. García-Castro. Benchmarking de la tecnología de la Web Semántica. In *Proceedings of the XII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA-TTIA2007), Doctoral Consortium*, Salamanca, Spain, November 2007.
- [García-Castro, 2007b] R. García-Castro. D6.8.1 Testing the NeOn Toolkit interoperability. Technical report, NeOn, September 2007.
- [Gediga *et al.*, 2002] G. Gediga, K. Hamborg, and I. Dutsch. *Evaluation of Software Systems*, volume Encyclopedia of Computer Science and Technology, Volume 44, pages 166–192. 2002.
- [Gee *et al.*, 2001] D. Gee, K. Jones, D. Kreitz, S. Nevell, B. O'Connor, and B. Van Ness. *Using Performance Information to Drive Improvement*, volume 6 of *The Performance-Based Management Handbook*. Performance-Based Management Special Interest Group, September 2001.
- [Gómez-Pérez *et al.*, 2003] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Springer Verlag, 2003.
- [Goodman, 1993] P. Goodman. *Practical Implementation of Software Metrics*. McGraw Hill, London, 1993.
- [Grady and Caswell, 1987] R.B. Grady and D.L. Caswell. *Software Metrics: Establishing a Company-Wide Program*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.
- [Grant and Beckett, 2004] J. Grant and D. Beckett. RDF test cases. Technical report, W3C, February 2004.
- [Gruber, 1993] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5, (2):199–220, 1993.
- [Guo *et al.*, 2003] Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL repositories. In *Proceedings of the 2nd International Semantic Web Conference, (ISWC2003)*, Florida, USA, October 2003.

- [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.
- [Hammer and McLeod, 1993] J. Hammer and D. McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Journal for Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.
- [Huang *et al.*, 2007] Z. Huang, J. Volker, Q. Ji, H. Stuckenschmidt, C. Meilicke, S. Schlobach, F. van Harmelen, and J. Lam. D1.2.2.1.4 Benchmarking of Processing Inconsistent Ontologies. Technical report, Knowledge Web, December 2007.
- [IEEE-STD-610, 1991] IEEE-STD-610. *ANSI/IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology*. IEEE, February 1991.
- [IEEE, 1998] IEEE. *IEEE Std 829-1998. IEEE Standard for Software Test Documentation*. IEEE, September 1998.
- [IEEE, 2000] IEEE. *IEEE 100. The Authoritative Dictionary of IEEE Standard Terms. Seventh edition*. IEEE, December 2000.
- [Isaac *et al.*, 2003] A. Isaac, R. Troncy, and V. Malais. Using XSLT for interoperability: DOE and the travelling domain experiment. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003.
- [ISO/IEC, 1999] ISO/IEC. *ISO/IEC 14598-1: Software product evaluation - Part 1: General overview*. 1999.
- [ISO/IEC, 2001] ISO/IEC. *ISO/IEC 9126-1. Software Engineering – Product Quality – Part 1: Quality model*. 2001.
- [Kim and Seo, 1991] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.
- [Kitchenham *et al.*, 2002] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El-Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28, (8):721–734, 2002.
- [Kitchenham, 1996] B. Kitchenham. DESMET: A method for evaluating software engineering methods and tools. Technical Report TR96-09, Department of Computer Science, University of Keele, Staffordshire, UK, 1996.
- [Klein, 2001] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In *Proceedings of the Workshop on Ontologies and Information Sharing (IJCAI2001)*, Seattle, USA, 2001.

- [Knublauch *et al.*, 2004] H. Knublauch, R.W. Fergerson, N.F. Noy, and M.A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, volume 3298, pages 229–243. Springer, 2004.
- [Knublauch, 2003a] H. Knublauch. Case study: Using Protégé to convert the travel ontology to UML and OWL. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003.
- [Knublauch, 2003b] H. Knublauch. Editing Semantic Web content with Protégé: the OWL plugin. In *Sixth Protégé Workshop*, Manchester, UK, 7-9 July 2003.
- [Kraft, 1997] J. Kraft. The Department of the Navy Benchmarking Handbook: A Systems View. Technical report, Department of the Navy, 1997.
- [Lankford, 2000] W.M. Lankford. Benchmarking: Understanding the basics. *Coastal Business Journal*, (1), 2000.
- [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 (ESWC2006)*, volume 4011 of *LNCS*, pages 125–139, Budva, Montenegro, June 11-14 2006.
- [Maynard *et al.*, 2007] D. Maynard, S. Dasiopolou, S. Costache, K. Eckert, H. Stuckenschmidt, M. Dzbor, and S Handschuh. D1.2.2.1.3 Benchmarking of annotation tools. Technical report, KnowledgeWeb, September 2007.
- [McAndrews, 1993] D.R. McAndrews. Establishing a software measurement process. Technical Report CMU/SEI-93-TR-16, SEI, July 1993.
- [McBride, 2001] B. McBride. Jena: Implementing the RDF Model and Syntax Specification. In *Proceedings of the Second International Workshop on the Semantic Web (SemWeb2001)*, May 2001.
- [Motik and Sattler, 2006] B. Motik and U. Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR2006)*, Phnom Penh, Cambodia, November 2006.
- [Motik *et al.*, 2002] B. Motik, A. Maedche, and R. Volz. A conceptual modeling approach for semantics-driven enterprise applications. In *Proceedings of the 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE2002)*, 2002.
- [Myers *et al.*, 2004] G.J. Myers, C. Sandler, T. Badgett, and T.M. Thomas. *The Art of Software Testing, Second Edition*. Wiley, June 2004.

- [Noy *et al.*, 2000] N. Noy, R. Fergerson, and M. Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. In *Proceedings of the 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW2000)*, Juan-les-Pins, France, 2000.
- [OntoWeb, 2002] OntoWeb. OntoWeb Deliverable 1.3: A survey on ontology tools. Technical report, OntoWeb Thematic Network, May 2002.
- [Paradela, 2001] L.F. Paradela. *Una metodología para la gestión de conocimientos*. PhD thesis, Facultad de Informática. Universidad Politécnica de Madrid, 2001.
- [Park *et al.*, 1996] R.E. Park, W.B. Goethert, and W.A. Florac. Goal-Driven Software Measurement - A Guidebook. Technical Report CMU/SEI-96-HB-002, Software Engineering Institute, August 1996.
- [Pfleeger, 1995] S.L. Pfleeger. Experimental design and analysis in software engineering, part 2: How to set up and experiment. *ACM SIGSOFT Software Engineering Notes*, 20(1):22–26, January 1995.
- [Rakitin, 1997] S. R. Rakitin. *Software Verification and Validation, A Practitioner's Guide*. Artech House, 1997.
- [Sheth, 1998] A. Sheth. *Interoperating Geographic Information Systems*, chapter Changing focus on interoperability in information systems: From system, syntax, structur to semantics, pages 5–30. Kluwer, 1998.
- [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 *Proceedings of the 11th 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 (ICSE2003)*, pages 74–83, Portland, OR, 2003.
- [Sole and Bist, 1995] T.D. Sole and G. Bist. Benchmarking in technical information. *IEEE Transactions on Professional Communication*, 38(2):77–82, June 1995.
- [Sommerville, 2006] Ian Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [Spendolini, 1992] M.J. Spendolini. *The Benchmarking Book*. AMACOM, New York, NY, 1992.
- [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.

- [Sure and Corcho, 2003] Y. Sure and O. Corcho, editors. *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, volume 87 of *CEUR-WS*, Florida, USA, October 2003.
- [Sure *et al.*, 2004] Y. Sure, A. Gómez-Pérez, W. Daelemans, M.-L. Reinberger, N. Guarino, and N.F. Noy. Why evaluate ontology technologies? because it works! *IEEE Intelligent Systems*, 19(4):74–81, July 2004.
- [Tamma, 2001] V. Tamma. *An ontology model supporting multiple ontologies for knowledge sharing*. PhD thesis, University of Liverpool, 2001.
- [Visser *et al.*, 1997] P.R.S. Visser, D.M. Jones, T.J.M. Bench-Capon, and M.J.R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.
- [Volz, 2004] Rapahel Volz. *Web ontology reasoning with logic databases*. PhD thesis, AIFB Karlsruhe, 2004.
- [Weiss, 2002] A.R. Weiss. Dhrystone benchmark: History, analysis, scores and recommendations. White paper, EEMBC Certification Laboratories, LLC, 2002.
- [Weithöner *et al.*, 2007] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F.W. von Henke, and O. Noppens. Real-world reasoning with OWL. In *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2007.
- [Wielemaker *et al.*, 2008] J. Wielemaker, Z. Huang, and L. van der Meij. SWI-Prolog and the Web. *Theory and Practice of Logic Programming*, pages 1–30, 2008.
- [Wireman, 2003] Terry Wireman. *Benchmarking Best Practices in Maintenance Management*. Industrial Press, 2003.
- [Wohlin *et al.*, 2000] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*, volume 6 of *International Series in Software Engineering*. Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A, 2000.
- [Wohlin *et al.*, 2002] C. Wohlin, A. Aurum, H. Petersson, F. Shull, and M. Ciolkowski. Software inspection benchmarking - a qualitative and quantitative comparative opportunity. In *Proceedings of 8th International Software Metrics Symposium*, pages 118–130, June 2002.

Combinations of the RDF(S) components

The method presented in this appendix has been devised to identify benchmarks that cover all the possible combinations of the RDF(S) knowledge model components.

Figure A.1 shows the different components that form the RDF(S) knowledge model and the different properties that relate them. In the figure, classes are defined as boxes, whereas properties are defined as arrows with their domain and range represented by the origin and destination of the arrow respectively. The components shown are instances of *rdfs:Class*, and some of them have predefined instances that do not appear here, since a full description of the knowledge model is not provided.

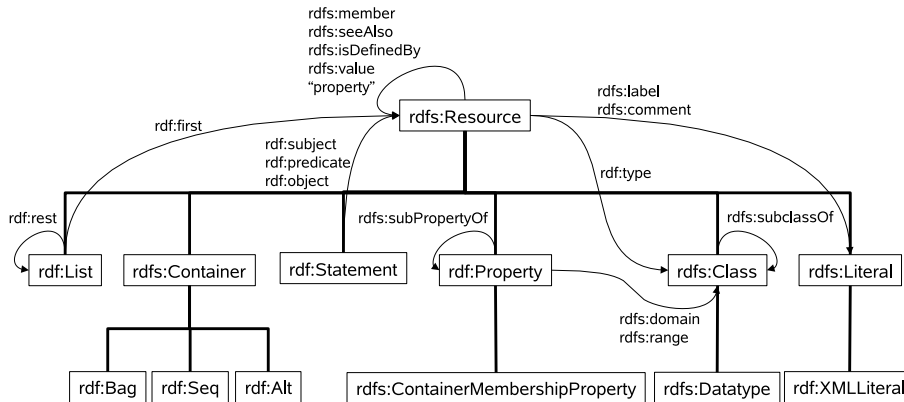


Figure A.1: The components of the RDF(S) knowledge model.

This method involves the following three different types of benchmarks:

- Benchmarks that import single components.

- Benchmarks that import all the possible combinations of two components with a property.
- Benchmarks that import combinations of more than two components usually appearing together in RDF(S) graphs.

A.1. Benchmarks with single components

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

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

A.2. Benchmarks with combinations of two components

To define all the combinations of two components related through a property, the following steps were executed on each of the RDF(S) components:

Step 1. To identify the possible relations of the component with others, i.e., the properties whose domain can be the component and relate this component to other components. These properties are

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

Step 2. To identify for each of these relations the ranges that the property can have and to assign the cardinalities that correspond to each relation. These ranges are

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

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

- 1:1 cardinality ($c1 \overset{1}{-} p \overset{1}{\rightarrow} c2$). To define 1 benchmark to import

- One instance of a component related to an instance of another component through a property.
- 1:N cardinality ($c1 \overset{1}{-} p \rightarrow^* c2$). To define 2 benchmarks, the one defined for the 1:1 cardinality and another to import
 - One instance of a component related to several instances of another component through the same property.
- N:1 cardinality ($c1 \overset{*}{-} p \rightarrow^1 c2$). To define 2 benchmarks, the one defined for the 1:1 cardinality and another to import
 - Several instances of a component related to an instance of another component through the same property.
- N:N cardinality ($c1 \overset{*}{-} p \rightarrow^* c2$). To define 3 benchmarks, the three defined for the 1:N and N:1 cardinalities.
- N:N cardinality, being c1 and c2 the same component ($c1 \overset{*}{-} p \rightarrow^* c1$). To define 4 benchmarks, three defined for the N:N cardinality and one to import
 - One instance of a component related to itself through a property.
- N:N cardinality, being c1 and c2 the same component and p a transitive property ($c1 \overset{*}{-} p \rightarrow^* c1$). To define 5 benchmarks, the four defined for the previous case and one to import
 - One instance of a component related to an instance of another component through a property, being the second instance related to an instance of a third component with the same property.

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

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

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

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

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

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

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

A.3. Benchmarks with combinations of more than two components

The main combinations of RDF(S) components that concern more than two components related through properties are the following:

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

The method to define the benchmarks is similar to the one described in the previous section. The main difference lies in the number of benchmarks defined according to the cardinalities. To clarify the explanation shown below, a cardinality of 1 is specified in the origin of the relation.

- If the cardinalities of the relations are

$$c1 \stackrel{1}{-} p1 \rightarrow^+ c2$$

$$c1 \stackrel{1}{-} p2 \rightarrow^+ c3$$
 then four benchmarks have been defined, which import:

- One instance of a component related to an instance of another component through a property and related to an instance of a third component through another property.
- One instance of a component related to several instances of another component through a property and related to an instance of another component through another property.
- One instance of a component related to an instance of another component through a property and related to several instances of another component through another property.
- One instance of a component related to several instances of another component through a property and related to several instances of another component through another property.

If *c2* and *c3* are the same component, then an additional benchmark has been defined, which imports:

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

- If the cardinalities of the relations are
 $c1 \xrightarrow{1-} p1 \xrightarrow{1} c2$
 $c1 \xrightarrow{1-} p2 \xrightarrow{1} c3$
 then one benchmark has been defined, which imports:

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

For example, for a property with domain and range and with the following ranges and cardinalities:

$rdf:Property \xrightarrow{1-} rdfs:domain \xrightarrow{+} rdfs:Class$
 $rdf:Property \xrightarrow{1-} rdfs:range \xrightarrow{+} rdfs:Class$

5 benchmarks have been defined, which import:

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

Description of the RDF(S) benchmark suites

B.1. RDF(S) Import Benchmark Suite

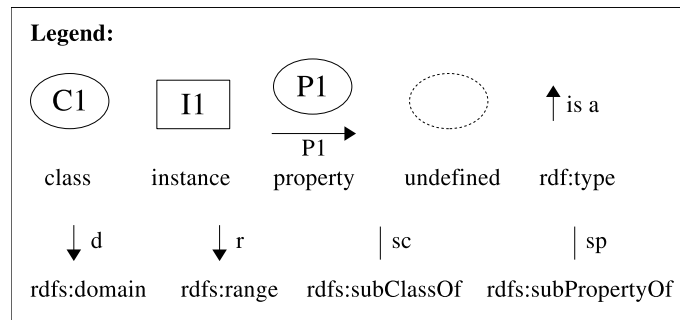


Figure B.1: Notation used in the RDF(S) Import Benchmark Suite figures.

Class benchmarks		
I01	Import just one class	C 1
I02	Import several classes with no proper- ties between them	C 1 C 2 C 3
Metaclass benchmarks		

(continued on next page)

(continued from previous page)

I03	Import one class that is instance of another class, being this last class instance of a third one	<pre> graph BT C3((C3)) -- "is a" --> C2((C2)) C2 -- "is a" --> C1((C1)) </pre>
I04	Import one class that is instance of several classes	<pre> graph BT C2((C2)) -- "is a" --> C1((C1)) C3((C3)) -- "is a" --> C1 </pre>
I05	Import several classes that are instance of the same class	<pre> graph BT C2((C2)) -- "is a" --> C1((C1)) C3((C3)) -- "is a" --> C1 </pre>
I06	Import one class that is instance of another class and viceversa	<pre> graph BT C1((C1)) -- "is a" --> C2((C2)) C2 -- "is a" --> C1 </pre>
I07	Import just one class that is instance of himself	<pre> graph BT C1((C1)) -- "is a" --> C1 </pre>
Subclass benchmarks		

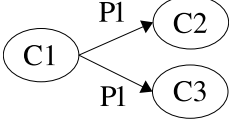
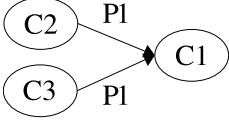
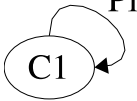
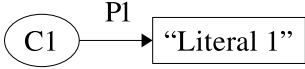
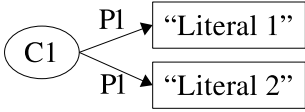
(continued on next page)

(continued from previous page)

I08	Import one class that is subclass of another class, being this last class subclass of a third one	<pre> graph BT C3((C3)) -- sc --> C2((C2)) C2 -- sc --> C1((C1)) </pre>
I09	Import one class that is subclass of several classes	<pre> graph BT C2((C2)) -- sc --> C1((C1)) C3((C3)) -- sc --> C1 </pre>
I10	Import several classes that are subclass of the same class	<pre> graph BT C2((C2)) -- sc --> C1((C1)) C3((C3)) -- sc --> C1 </pre>
I11	Import one class that is subclass of another class and viceversa, forming a cycle	<pre> graph TD C1((C1)) -- sc --> C2((C2)) C2 -- sc --> C1 </pre>
I12	Import just one class that is subclass of himself, forming a cycle	<pre> graph TD C1((C1)) -- sc --> C1 </pre>
Class and property benchmarks		
I13	Import one class that has a property with another class. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	<pre> graph LR C1((C1)) -- P1 --> C2((C2)) </pre>

(continued on next page)

(continued from previous page)

I14	Import one class that has the same property with several classes. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	
I15	Import several classes that have the same property with the same class. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	
I16	Import just one class that has a property with itself. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	
I17	Import just one class that has a property with a literal. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	
I18	Import just one class that has the same property with several literals. The property is supposed to be defined with a domain and a range of some metaclass of the classes (such as <code>rdfs:Class</code>)	
Single property benchmarks		
I19	Import just one property	
I20	Import several properties	
Subproperty benchmarks		

(continued on next page)

(continued from previous page)

I21	Import one property that is subproperty of another property that is subproperty of a third one	<pre> graph BT P3((P3)) -- sp --> P2((P2)) P2 -- sp --> P1((P1)) </pre>
I22	Import one property that is subproperty of several properties	<pre> graph BT P2((P2)) -- sp --> P1((P1)) P3((P3)) -- sp --> P1 </pre>
I23	Import several properties that are subproperty of the same property	<pre> graph TD P1((P1)) -- sp --> P2((P2)) P1 -- sp --> P3((P3)) </pre>
I24	Import one property that is subproperty of another property and viceversa	<pre> graph TD P2((P2)) -- sp --> P1((P1)) P1 -- sp --> P2 </pre>
I25	Import just one property that is subproperty of himself	<pre> graph TD P1((P1)) -- sp --> P1 </pre>
Property with domain and range benchmarks		

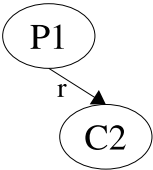
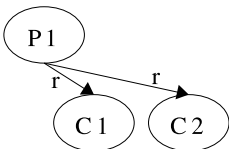
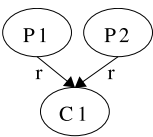
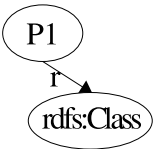
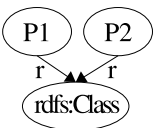
(continued on next page)

(continued from previous page)

I26	Import just one property that has as domain a resource and as range another resource, without the resource definitions	
I27	Import just one property that has as domain a class, with the class defined in the ontology	
I28	Import just one property that has as domain several classes, with the classes defined in the ontology	
I29	Import several properties that have as domain the same class, with the class defined in the ontology	
I30	Import just one property that has as domain rdfs:Class	
I31	Import several properties that have as domain rdfs:Class	

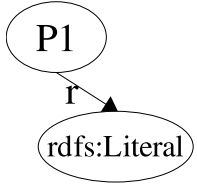
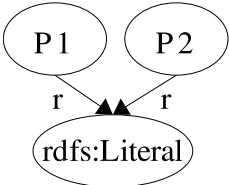
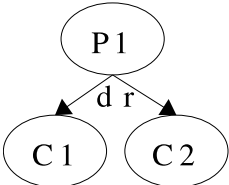
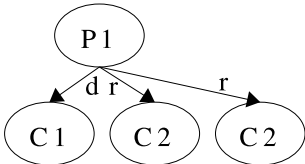
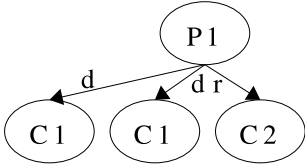
(continued on next page)

(continued from previous page)

I32	Import just one property that has as range a class, with the class defined in the ontology	 <pre> graph TD P1((P1)) -- r --> C2((C2)) </pre>
I33	Import just one property that has as range several classes, with the classes defined in the ontology	 <pre> graph TD P1((P1)) -- r --> C1((C1)) P1 -- r --> C2((C2)) </pre>
I34	Import several properties that have as range the same class, with the class defined in the ontology	 <pre> graph TD P1((P1)) -- r --> C1((C1)) P2((P2)) -- r --> C1 </pre>
I35	Import just one property that has as range <code>rdfs:Class</code>	 <pre> graph TD P1((P1)) -- r --> rdfsClass([rdfs:Class]) </pre>
I36	Import several properties that have as range <code>rdfs:Class</code>	 <pre> graph TD P1((P1)) -- r --> rdfsClass([rdfs:Class]) P2((P2)) -- r --> rdfsClass </pre>

(continued on next page)

(continued from previous page)

I37	Import just one property that has as range <code>rdfs:Literal</code>	 <pre> graph TD P1((P1)) -- r --> Literal(rdfs:Literal) </pre>
I38	Import several properties that have as range <code>rdfs:Literal</code>	 <pre> graph TD P1((P1)) -- r --> Literal(rdfs:Literal) P2((P2)) -- r --> Literal </pre>
I39	Import just one property that has as domain a class and as range another class, with the classes defined in the ontology	 <pre> graph TD P1((P1)) -- d r --> C1((C1)) P1 -- d r --> C2((C2)) </pre>
I40	Import just one property that has as domain a class and as range several classes, with the classes defined in the ontology	 <pre> graph TD P1((P1)) -- d r --> C1((C1)) P1 -- d r --> C2a((C2)) P1 -- r --> C2b((C2)) </pre>
I41	Import just one property that has as domain several classes and as range a class, with the classes defined in the ontology	 <pre> graph TD P1((P1)) -- d r --> C1a((C1)) P1 -- d r --> C2((C2)) P1 -- d --> C1b((C1)) </pre>

(continued on next page)

(continued from previous page)

I42	Import just one property that has as domain several classes and as range several classes, with the classes defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1_1((C1)) P1 -- d --> C1_2((C1)) P1 -- r --> C2_1((C2)) P1 -- r --> C2_2((C2)) </pre>
I43	Import just one property that has as domain and range the same class, with the class defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> C1 </pre>
I44	Import just one property that has as domain a class and as range rdfs:Literal, with the class defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> Literal(rdfs:Literal) </pre>
I45	Import just one property that has as domain several classes and as range rdfs:Literal, with the classes defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- d --> C2((C2)) P1 -- r --> Literal(rdfs:Literal) </pre>
I46	Import just one property that has as domain a class and as range the XML Schema datatype "string", with the class defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> String(xsd:string) </pre>
I47	Import just one property that has as domain several classes and as range the XML Schema datatype "integer", with the classes defined in the ontology	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- d --> C2((C2)) P1 -- r --> Integer(xsd:integer) </pre>

(continued on next page)

(continued from previous page)

I48	Import just one property that has as domain <code>rdfs:Class</code> and as range <code>rdfs:Class</code>	
I49	Import just one property that has as domain <code>rdfs:Class</code> and as range <code>rdfs:Literal</code>	
Instance benchmarks		
I50	Import just one instance of a resource, without the resource definition	
I51	Import one class and one instance of the class	
I52	Import several classes and one instance of all of them	

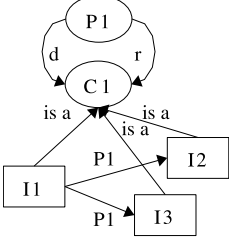
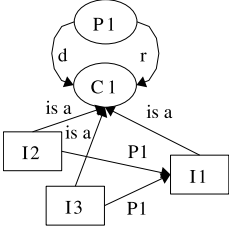
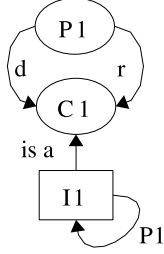
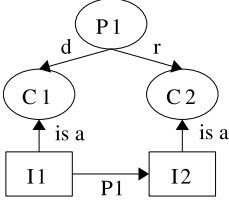
(continued on next page)

(continued from previous page)

I53	Import one class and several instances of the class	
Instance and property benchmarks		
I54	Import one class and one instance of the class that has a property with another instance of the same class, without the property definition	
I55	Import two classes and one instance of one class that has a property with an instance of the other class, without the property definition	
I56	Import one class and one instance of the class that has a property with a literal, without the property definition	
I57	Import one class, one property with domain and range the class, and one instance of the class that has the property with another instance of the same class	

(continued on next page)

(continued from previous page)

I58	Import one class, one property with domain and range the class, and one instance of the class that has the property with several instances of the class	
I59	Import one class, one property with domain and range the class, and several instances of the class that have the property with the same instance of the class	
I60	Import one class, one property with domain and range the class, and one instance of the class that has the property with himself	
I61	Import two classes, one property with domain one class and range the other class, and one instance of one class that has the property with an instance of the other class	

(continued on next page)

(continued from previous page)

I62	Import two classes, one property with domain one class and range the other class, and one instance of one class that has the property with several instances of the other class	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> C2((C2)) C1 -- is a --> I1[I1] C2 -- is a --> I2[I2] C2 -- is a --> I3[I3] I1 -- P1 --> I2 I1 -- P1 --> I3 I2 -- P1 --> I3 </pre>
I63	Import two classes, one property with domain one class and range the other class, and several instances of one class that have the property with the same instance of the other class	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> C2((C2)) C1 -- is a --> I2[I2] C1 -- is a --> I3[I3] C2 -- is a --> I1[I1] I2 -- P1 --> I1 I3 -- P1 --> I1 </pre>
I64	Import one class, one property with domain the class and range rdfs:Literal, and one instance of the class that has the property with a literal	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> Literal1((rdfs:Literal)) C1 -- is a --> I1[I1] I1 -- P1 --> Literal1 </pre>
I65	Import one class, one property with domain the class and range rdfs:Literal, and one instance of the class that has the property with several literals	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> Literal1((rdfs:Literal)) P1 -- r --> Literal2((rdfs:Literal)) C1 -- is a --> I1[I1] I1 -- P1 --> Literal1 I1 -- P1 --> Literal2 </pre>

(continued on next page)

(continued from previous page)

I66	Import one class, one property with domain the class and range the XML Schema datatype “string”, and one instance of the class that has the property with a value	<pre> graph TD P1((P1)) -- d r --> C1((C1)) P1 -- d r --> xsd_string((xsd:string)) C1 -- is a --> I1[I1] xsd_string -- is a --> Literal1[Literal 1] I1 -- P1 --> Literal1 </pre>
I67	Import one class, one property with domain the class and range the XML Schema datatype “integer”, and one instance of the class that has the property with several integer values	<pre> graph TD P1((P1)) -- d r --> C1((C1)) P1 -- d r --> xsd_integer((xsd:integer)) C1 -- is a --> I1[I1] xsd_integer -- is a --> 10["10"] xsd_integer -- is a --> 11["11"] I1 -- P1 --> 10 I1 -- P1 --> 11 </pre>
Syntax and abbreviation benchmarks		
URI reference benchmarks		
I68	Import several resources with absolute URI references	
I69	Import several resources with URI references relative to a base URI	
I70	Import several resources with URI references transformed from rdf:ID attribute values	
I71	Import several resources with URI references relative to an ENTITY declaration	
Empty node benchmarks		
I72	Import several resources with empty nodes	
I73	Import several resources with empty nodes shortened	
Multiple properties benchmarks		
I74	Import several resources with multiple properties	
I75	Import several resources with multiple properties shortened	
Typed node benchmarks		

(continued on next page)

(continued from previous page)

I76	Import several resources with typed nodes	
I77	Import several resources with typed nodes shortened	
String literal benchmarks		
I78	Import several resources with properties with string literals	
I79	Import several resources with properties with string literals as XML attributes	
Blank node benchmarks		
I80	Import several resources with blank nodes with identifier	
I81	Import several resources with blank nodes shortened	
Language identification benchmarks		
I82	Import several resources with properties with xml:lang attributes	

B.2. RDF(S) Export and Interoperability Benchmark Suites

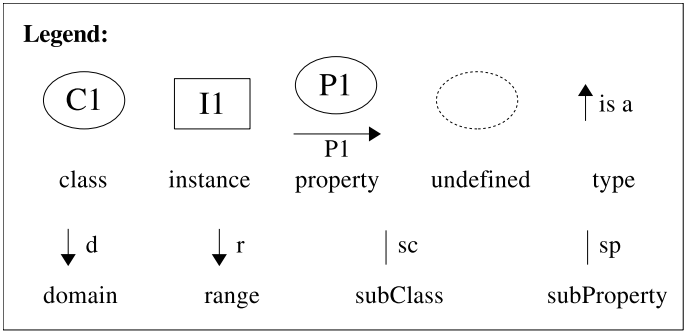
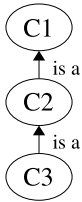
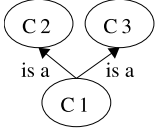
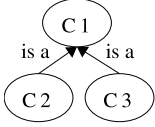
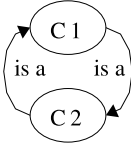
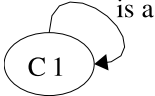


Figure B.2: Notation used in the RDF(S) Export Benchmark Suite figures.

Class benchmarks		
E01	Export just one class	C 1

(continued on next page)

(continued from previous page)

E02	Export several classes	
Metaclass benchmarks		
E03	Export one class that is instance of another class that is instance of a third one	
E04	Export one class that is instance of several classes	
E05	Export several classes that are instance of the same class	
E06	Export one class that is instance of another class and viceversa	
E07	Export just one class that is instance of himself	
Subclass benchmarks		

(continued on next page)

(continued from previous page)

E08	Export one class that is subclass of another class that is subclass of a third one	<pre> graph BT C3((C3)) -- sc --> C2((C2)) C2 -- sc --> C1((C1)) </pre>
E09	Export one class that is subclass of several classes	<pre> graph BT C2((C2)) -- sc --> C1((C1)) C3((C3)) -- sc --> C1 </pre>
E10	Several classes that are subclass of the same class	<pre> graph BT C2((C2)) -- sc --> C1((C1)) C3((C3)) -- sc --> C1 </pre>
E11	Export one class that is subclass of another class and viceversa, forming a cycle	<pre> graph TD C1((C1)) -- sc --> C2((C2)) C2 -- sc --> C1 </pre>
E12	Export just one class that is subclass of himself, forming a cycle	<pre> graph TD C1((C1)) -- sc --> C1 </pre>
Class and object property benchmarks		
E13	Export one class that has an object property with another class. The property is supposed to be defined with a domain and a range of some metaclass of the classes	<pre> graph LR C1((C1)) -- P1 --> C2((C2)) </pre>

(continued on next page)

(continued from previous page)

E14	Export one class that has the same object property with several classes. The property is supposed to be defined with a domain and a range of some metaclass of the classes	
E15	Export several classes that have the same object property with the same class. The property is supposed to be defined with a domain and a range of some metaclass of the classes	
E16	Export just one class that has an object property with itself. The property is supposed to be defined with a domain and a range of some metaclass of the class	
Class and datatype property benchmarks		
E17	Export just one class that has a datatype property with a literal. The property is supposed to be defined with a domain and a range of some metaclass of the class	
E18	Export just one class that has the same datatype property with several literals. The property is supposed to be defined with a domain and a range of some metaclass of the class	
Datatype property benchmarks		
E19	Export just one datatype property	
E20	Export several datatype properties	

(continued on next page)

(continued from previous page)

E21	Export just one datatype property that has as domain a resource and as range “String”, without the resource definition	
E22	Export just one datatype property that has as domain a class, with the class defined in the ontology	
E23	Export just one datatype property that has as domain several classes, with the classes defined in the ontology	
E24	Export several datatype properties that have as domain the same class, with the class defined in the ontology	
E25	Export just one datatype property that has as range “String”	
E26	Export several datatype properties that have as range “String”	

(continued on next page)

(continued from previous page)

E27	Export one datatype property that has as domain a class and as range “String”, with the class defined in the ontology	
E28	Export one datatype property that has as domain several classes and as range “String”, with the classes defined in the ontology	
E29	Export one datatype property that has as domain a class and as range the XML Schema datatype “string”, with the class defined in the ontology	
E30	Export one datatype property that has as domain several classes and as range the XML Schema datatype “integer”, with the classes defined in the ontology	
Object property benchmarks		
E31	Export just one object property	
E32	Export several object properties	

(continued on next page)

(continued from previous page)

E33	Export just one object property that has as domain a resource and as range another resource, without the resource definitions	
E34	Export just one object property that has as domain a class, with the class defined in the ontology	
E35	Export just one object property that has as domain several classes, with the classes defined in the ontology	
E36	Export several object properties that have as domain the same class, with the class defined in the ontology	
E37	Export just one object property that has as range a class, with the class defined in the ontology	

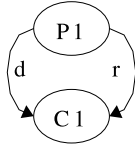
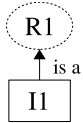
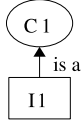
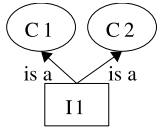
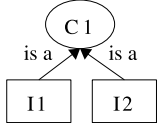
(continued on next page)

(continued from previous page)

E38	Export just one object property that has as range several classes, with the classes defined in the ontology	
E39	Export several object properties that have as range the same class, with the class defined in the ontology	
E40	Export just one object property that has as domain a class and as range another class, with the classes defined in the ontology	
E41	Export just one object property that has as domain a class and as range several classes, with the classes defined in the ontology	
E42	Export just one object property that has as domain several classes and as range a class, with the classes defined in the ontology	
E43	Export just one object property that has as domain several classes and as range several classes, with the classes defined in the ontology	

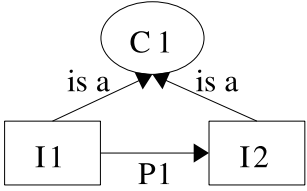
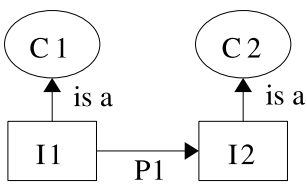
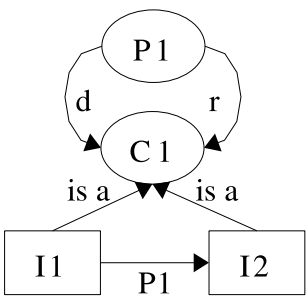
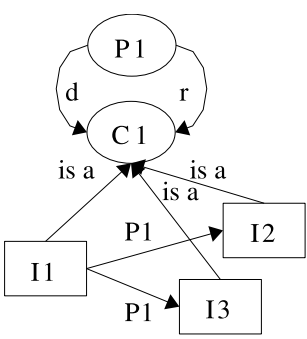
(continued on next page)

(continued from previous page)

E44	Export just one object property that has as domain and range the same class, with the class defined in the ontology	
Instance benchmarks		
E45	Export just one instance of a resource, without the resource definition	
E46	Export one class and one instance of the class	
E47	Export several classes and one instance of all of them	
E48	Export one class and several instances of the class	
Instance and object property benchmarks		

(continued on next page)

(continued from previous page)

E49	Export one class and one instance of the class that has an object property with another instance of the same class, without the property definition	
E50	Export two classes and one instance of one class that has an object property with an instance of the other class, without the property definition	
E51	Export one class, one object property with domain and range the class, and one instance of the class that has the property with another instance of the same class	
E52	Export one class, one object property with domain and range the class, and one instance of the class that has the property with several instances of the class	

(continued on next page)

(continued from previous page)

E53	Export one class, one object property with domain and range the class, and several instances of the class that have the property with the same instance of the class	
E54	Export one class, one object property with domain and range the class, and one instance of the class that has the property with himself	
E55	Export two classes, one object property with domain one class and range the other class, and one instance of one class that has the property with an instance of the other class	

(continued on next page)

(continued from previous page)

E56	Export two classes, one object property with domain one class and range the other class, and one instance of one class that has the property with several instances of the other class	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> C2((C2)) I1[I1] -- is a --> C1 I2[I2] -- is a --> C2 I3[I3] -- is a --> C2 I1 -- P1 --> I2 I1 -- P1 --> I3 </pre>
E57	Export two classes, one object property with domain one class and range the other class, and several instances of one class that have the property with the same instance of the other class	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> C2((C2)) I2[I2] -- is a --> C1 I3[I3] -- is a --> C1 I1[I1] -- is a --> C2 I2 -- P1 --> I1 I3 -- P1 --> I1 </pre>
Instance and datatype property benchmarks		
E58	Export one class and one instance of the class that has a datatype property with a literal, without the property definition	<pre> graph TD C1((C1)) I1[I1] -- is a --> C1 I1 -- P1 --> L1["Literal 1"] </pre>
E59	Export one class, one datatype property with domain the class and range "String", and one instance of the class that has the property with a literal	<pre> graph TD P1((P1)) -- d --> C1((C1)) P1 -- r --> String((String)) I1[I1] -- is a --> C1 I1 -- P1 --> L1["Literal 1"] L1 -- is a --> String </pre>

(continued on next page)

(continued from previous page)

E60	Export one class, one datatype property with domain the class and range “String”, and one instance of the class that has the property with several literals	<pre> graph TD P1((P1)) -- d r --> C1((C1)) P1((P1)) -- d r --> String(String) C1((C1)) -- is a --> I1[I1] String(String) -- is a --> L1["Literal 1"] String(String) -- is a --> L2["Literal 2"] I1[I1] -- P1 --> L1 I1[I1] -- P1 --> L2 </pre>
E61	Export one class, one datatype property with domain the class and range the XML Schema datatype “string”, and one instance of the class that has the property with a value	<pre> graph TD P1((P1)) -- d r --> C1((C1)) P1((P1)) -- d r --> xsd_string(xsd:string) C1((C1)) -- is a --> I1[I1] xsd_string(xsd:string) -- is a --> L1["Literal 1"] I1[I1] -- P1 --> L1 </pre>
E62	Export one class, one datatype property with domain the class and range the XML Schema datatype “integer”, and one instance of the class that has the property with several integer values	<pre> graph TD P1((P1)) -- d r --> C1((C1)) P1((P1)) -- d r --> xsd_integer(xsd:integer) C1((C1)) -- is a --> I1[I1] xsd_integer(xsd:integer) -- is a --> L1["10"] xsd_integer(xsd:integer) -- is a --> L2["11"] I1[I1] -- P1 --> L1 I1[I1] -- P1 --> L2 </pre>
URI character restrictions		
Concepts and properties whose names start with a character that is not a letter or ‘_’		
E63	Export an ontology containing two classes named “1class” and “2class”, each with one datatype property of type String named “-datatypeProperty1” and “-datatypeProperty2” respectively, and an object property between the classes named “.objectProperty”	
Concepts and properties with spaces in their names		

(continued on next page)

(continued from previous page)

E64	Export an ontology containing two classes named “class 1” and “class 2”, each with one datatype property of type String named “datatype property 1” and “datatype property 2” respectively, and an object property between the classes named “object property”	
Concepts and properties with URI reserved characters in their names (’, ’/’, ’?’, ’:’, ’@’, ’&’, ’=’, ’+’, ’\$’, ’,’)		
E65	Export an ontology containing two classes named “concept/1” and “concept:1”, each with one datatype property of type String named “datatype/property/1” and “datatype=property=2” respectively, and an object property between the classes named “object\$property”	
Concepts and properties with XML delimiter characters in their names (’, ’_’, ’#’, ’%’, ’“”)		
E66	Export an ontology containing two classes named “class_1” and “class_1”, each with one datatype property of type String named “datatype#property#1” and “datatype%property%2” respectively, and an object property between the classes named “object”property”	

Combinations of the OWL Lite components

The following method, jointly devised by the author and Stefano David, can be used to identify benchmarks that cover the combinations of the OWL Lite knowledge model components.

This method contemplates three different types of benchmarks, according to the main OWL components that they manage, which are

- Benchmarks for classes.
- Benchmarks for properties.
- Benchmarks instances.

The conventions used in the productions are those used in the OWL specification¹, i.e., the first character of a class 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.

C.1. Benchmarks for classes

In OWL Lite, classes can be described by a class identifier, a value or a cardinality restriction on a property, and the intersection operator. Using these building blocks, the OWL Lite class and restriction axioms were applied to define the different ways of describing a class in OWL Lite with these axioms.

Then, the benchmarks were grouped according to the following criteria: classes and class hierarchies, class equivalences, and classes defined with a set operator.

¹<http://www.w3.org/TR/owl-semantics/syntax.html>

Group A: Classes and class hierarchies

This group contains ontologies that describe classes and class hierarchies. These ontologies include classes that are a subclass of value restrictions, cardinality restrictions on properties, and class intersections.

In this group, vocabulary terms of both RDF(S) and OWL² are used:

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

The productions used for defining the benchmarks are

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

```
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 glance at 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 in turn can have only instances of class *Car* as range.

This ontology can be expressed in the OWL abstract syntax 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)))
)
```

²The main vocabulary terms of the group are highlighted in boldface.

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

This group contains ontologies that 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 similar to 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'*).

Therefore, the vocabulary terms used are

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

The productions 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

This group contains ontologies that describe classes 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.

Therefore, the vocabulary terms used are

```
owl:intersectionOf, owl:Class
```

The production used for defining these benchmarks are

```

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

```

C.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).

The benchmarks of this group were grouped 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

This group contains ontologies that describe properties and property hierarchies.

Therefore, the vocabulary terms used 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

This group contains ontologies that describe properties that have from one to three domain and/or range constraints. Properties with no range and domain constraint are not contemplated since they are already included in Group D.

Therefore, the vocabulary terms used 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

This group contains ontologies that describe equivalences among object properties and among datatype properties; they also describe object properties that are inverse. 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, which is not allowed in OWL Lite.

Therefore, the vocabulary terms used are

```

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

```

In this group the following axioms are used:

```

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. This group contains ontologies that 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.

Therefore, the vocabulary terms used are

```

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

```

The axiom employed 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

```

C.3. Benchmarks for instances

In OWL Lite, individuals (named or anonymous) are instances of classes related to other individuals through properties. Special built-in properties for asserting relationships among them can also be found in OWL Lite.

Group H: Single individuals

The easiest way to describe an individual is to instantiate a class. This group contains ontologies that define one or more classes with single or multiple

individuals as instances.

Thus the only vocabulary terms used are *owl:Class* and *rdf:type*. On the other hand, 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 to each other through user-defined properties. In this group, every ontology has one object or datatype property whose domain and range are classes, and has individuals as instance of these classes. Moreover, the object and datatype properties are simple (no logical characteristics of properties exist) and, in the case of datatype properties, there are also data values (only strings were used).

Therefore, the vocabulary terms used for defining classes and properties with range and domain constraints, and individuals that are instances of these classes and properties, are

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

The axioms used 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., they can be referred without having to give them an explicit name, but they can be used in assertions.

Therefore, the vocabulary terms used for defining classes and properties with range and domain constraints are

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

In this group, the OWL Lite axioms used 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 contains built-in predicates (i.e., terms) that express basic relations among individuals. These terms are 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.

Therefore, the vocabulary terms used for defining classes and properties with range and domain constraint are

```
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 used 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

```

It can be observed that there is not an explicit production for generating the vocabulary terms *owl:AllDifferent* and *owl:distinctMembers*. The abstract syntax of OWL allows producing only pairwise disjoint individuals, and these two 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.

The OWL Lite Import Benchmark Suite

This appendix contains a list of the 82 benchmarks that compose the OWL Lite Import Benchmark Suite and their description in DL.

D.1. List of benchmarks

The benchmarks that compose the OWL Lite Import Benchmark Suite are described by

- A unique identifier (i.e., **ISA01** where **IS** denotes the OWL Lite 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 D.2.
- A graphical representation of the ontology, which uses the notation shown in figure D.1

The benchmarks that compose the OWL Lite Import Benchmark Suite are defined in the following tables.

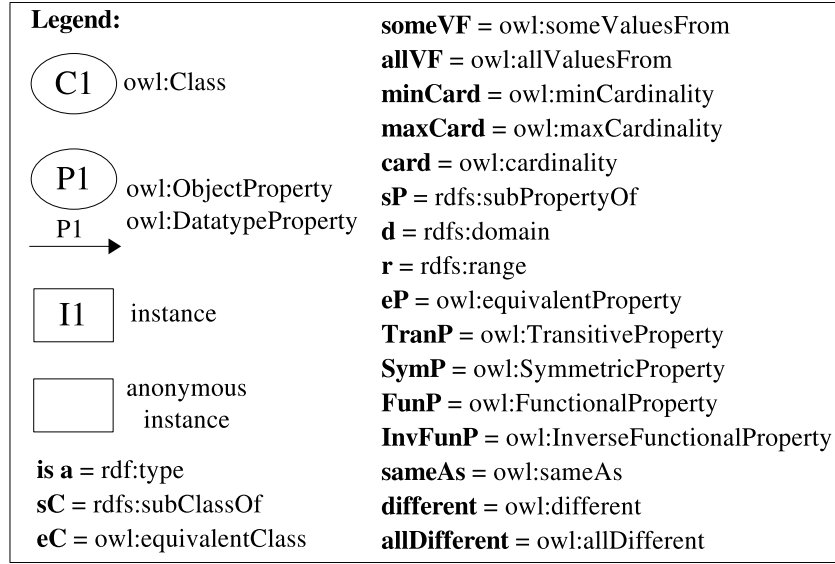
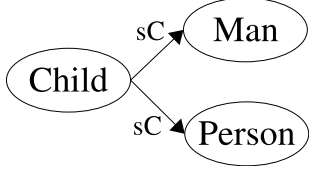


Figure D.1: Notation used in the OWL Lite Import Benchmark Suite figures.

Class benchmarks

Group A: Class hierarchies

ID	Description	Graphical representation
ISA01	Import a single class	
ISA02	Import a single class, subclass of a second class which is subclass of a third one	
ISA03	Import a class that is subclass of two classes	

(continued on next page)

(continued from previous page)

ISA04	Import several classes subclass of a single class	<pre> graph LR Man((Man)) -- sC --> Person((Person)) Woman((Woman)) -- sC --> Person </pre>
ISA05	Import two classes, each subclass of the other	<pre> graph LR Man((Man)) -- sC --> Male((Male)) Male -- sC --> Man </pre>
ISA06	Import a class, subclass of itself	<pre> graph LR Woman((Woman)) -- sC --> Woman </pre>
ISA07	Import a class which is subclass of an anonymous class defined by an owl:someValuesFrom value constraint in an object property	<pre> graph LR Castellano((Castellano)) -- sC --> AC1([wasBorn/ someVF(Spain)]) </pre>
ISA08	Import a class which is subclass of an anonymous class defined by an owl:allValuesFrom value constraint in an object property	<pre> graph LR Italian((Italian)) -- sC --> AC2([wasBorn/ allVF(Italy)]) </pre>
ISA09	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property	<pre> graph LR Employee((Employee)) -- sC --> AC3([worksIn/ minCard(0)]) </pre>

(continued on next page)

(continued from previous page)

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	

(continued on next page)

(continued from previous page)

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	
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	

(continued on next page)

(continued from previous page)

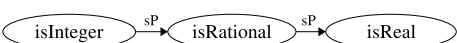
ISB10	Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property	<pre> graph LR Person([Person]) -- eC --> hasSSN([hasSSN/ card(1)]) </pre>
ISB11	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in a datatype property	<pre> graph LR Researcher([Researcher]) -- eC --> wrotePhD1([wrotePhDThesis/ minCard(0)]) Researcher -- eC --> wrotePhD2([wrotePhDThesis/ maxCard(1)]) </pre>
ISB12	Import a class which is equivalent to an anonymous class defined by the intersection of two other classes	<pre> graph LR ItalianMan([ItalianMan]) -- eC --> Intersection subgraph Intersection Italian([Italian]) Male([Male]) end </pre>

Group C: Classes defined with set operators

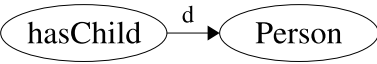
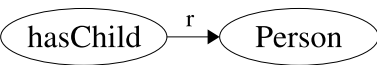
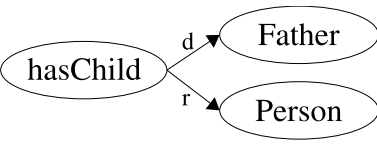
ID	Description	Graphical representation
ISC01	Import a class which is intersection of two other classes	<pre> graph LR ItalianMan([ItalianMan]) --> Intersection subgraph Intersection Italian([Italian]) Male([Male]) end </pre>
ISC02	Import a class which is intersection of several other classes	<pre> graph LR ItalianMan([ItalianMan]) --> Intersection subgraph Intersection Italian([Italian]) Male([Male]) Person([Person]) end </pre>

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	

(continued on next page)

(continued from previous page)

ISE04	Import a single object property with domain and range the same class	<pre> graph LR hasChild((hasChild)) Person((Person)) hasChild -- d --> Person hasChild -- r --> Person </pre>
ISE05	Import a single object property with domain multiple classes and range a class	<pre> graph LR hasChild((hasChild)) Woman((Woman)) Mother((Mother)) Person((Person)) hasChild -- d --> Woman hasChild -- d --> Mother hasChild -- d --> Person hasChild -- r --> Person </pre>
ISE06	Import a single object property with domain a class and range multiple classes	<pre> graph LR hasChild((hasChild)) Child((Child)) Human((Human)) Person((Person)) hasChild -- r --> Child hasChild -- r --> Human hasChild -- r --> Person hasChild -- d --> Person </pre>
ISE07	Import a single datatype property with domain a class	<pre> graph LR hasSSN((hasSSN)) Person((Person)) hasSSN -- d --> Person </pre>
ISE08	Import a single datatype property with range <code>rdfs:Literal</code>	<pre> graph LR hasName((hasName)) rdfsLiteral((rdfs:Literal)) hasName -- r --> rdfsLiteral </pre>
ISE09	Import a single datatype property with domain a class and range <code>rdfs:Literal</code>	<pre> graph LR hasName((hasName)) Person((Person)) rdfsLiteral((rdfs:Literal)) hasName -- d --> Person hasName -- r --> rdfsLiteral </pre>

(continued on next page)

(continued from previous page)

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
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	
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	
ISH02	Import several classes and one individual that is instance of all of them	
ISH03	Import one class and several individuals that are instance of the class	

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	

(continued on next page)

(continued from previous page)

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	
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	
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
Empty node benchmarks	

ISL05	Import several resources with empty nodes
ISL06	Import several resources with empty nodes shortened
Multiple properties benchmarks	
ISL07	Import several resources with multiple properties
ISL08	Import several resources with multiple properties shortened
Empty node benchmarks	
ISL09	Import several resources with typed nodes
ISL10	Import several resources with typed nodes shortened
Empty node benchmarks	
ISL11	Import several resources with properties with string literals
ISL12	Import several resources with properties with string literals as XML attributes
Empty node benchmarks	
ISL13	Import several resources with blank nodes with identifier
ISL14	Import several resources with blank nodes shortened
Language identification benchmarks	
ISL15	Import several resources with properties with xml:lang attributes

D.2. Description of ontologies in DL

This appendix provides a formal description of the ontologies that compose the OWL Lite Import Benchmark Suite in the Description Logics formalism.

The formalism presented in this appendix adopts the following conventional notation, shown in table D.1 and presented in [Volz, 2004], to map the OWL axioms in the abstract syntax to Description Logics concepts. On the left column it appears the abstract syntax of an OWL axiom and on the right column the corresponding axiom expressed in the Description Logics formalism.

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

Table D.1: Description Logics notation from [Volz, 2004].

Table D.2 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 D.2: Sample ontology description in the Description Logics formalism.

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
ISA07	Import a class which is subclass of an anonymous class defined by an owl:someValuesFrom value constraint in an object property Driver $\sqsubseteq$ $\exists hasCar.Car$
ISA08	Import a class which is subclass of an anonymous class defined by an owl:allValuesFrom value constraint in an object property Italian $\sqsubseteq$ $\forall wasBorn.Italy$
ISA09	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property Employee $\sqsubseteq$ $\geq 0 worksIn$
ISA10	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property Researcher $\sqsubseteq$ $\leq 1 hasAffiliation$
ISA11	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property Man $\sqsubseteq$ $= 1 hasMother$

(continued on next page)

(continued from previous page)

ISA12	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 an object property $\text{Researcher} \sqsubseteq \geq 0 \text{ hasAffiliation}$ $\text{Researcher} \sqsubseteq \leq 1 \text{ hasAffiliation}$
ISA13	Import a class which is subclass of an anonymous class defined by an <code>owl:minCardinality=0</code> cardinality constraint in a datatype property $\text{Person} \sqsubseteq \geq 0 \text{ hasName}$
ISA14	Import a class which is subclass of an anonymous class defined by an <code>owl:maxCardinality=1</code> cardinality constraint in a datatype property $\text{Researcher} \sqsubseteq \leq 1 \text{ wrotePhDThesis}$
ISA15	Import a class which is subclass of an anonymous class defined by an <code>owl:cardinality=1</code> cardinality constraint in a datatype property $\text{Person} \sqsubseteq = 1 \text{ hasSSN}$
ISA16	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 $\text{Researcher} \sqsubseteq \geq 0 \text{ wrotePhDThesis}$ $\text{Researcher} \sqsubseteq \leq 1 \text{ wrotePhDThesis}$
ISA17	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

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

(continued on next page)

(continued from previous page)

ISB05	Import a class which is equivalent to an anonymous class defined by an <code>owl:maxCardinality=1</code> cardinality constraint in an object property $\text{Researcher} \equiv \leq 1 \text{ hasAffiliation}$
ISB06	Import a class which is equivalent to an anonymous class defined by an <code>owl:cardinality=1</code> cardinality constraint in an object property $\text{Man} \equiv = 1 \text{ hasMother}$
ISB07	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 $\text{Researcher} \equiv (\leq 1 \text{ hasAffiliation} \sqcap \geq 0 \text{ hasAffiliation})$
ISB08	Import a class which is equivalent to an anonymous class defined by an <code>owl:minCardinality=0</code> cardinality constraint in a datatype property $\text{Person} \equiv \geq 0 \text{ hasName}$
ISB09	Import a class which is equivalent to an anonymous class defined by an <code>owl:maxCardinality=1</code> cardinality constraint in a datatype property $\text{Researcher} \equiv \leq 1 \text{ wrotePhDThesis}$
ISB10	Import a class which is equivalent to an anonymous class defined by an <code>owl:cardinality=1</code> cardinality constraint in a datatype property $\text{Person} \equiv = 1 \text{ hasSSN}$
ISB11	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 $\text{Researcher} \equiv \geq 0 \text{ wrotePhDThesis}$ $\text{Researcher} \equiv \leq 1 \text{ wrotePhDThesis}$
ISB12	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

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

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

Group E: Properties with domain and range

ISE01	Import a single object property with domain a class $\top \sqsubseteq \forall hasChild \top . Person$
ISE02	Import a single object property with range a class $\top \sqsubseteq \forall hasChild . Person$
ISE03	Import a single object property with domain a class and range another class $\top \sqsubseteq \forall hasChild \top . Father$ $\top \sqsubseteq \forall hasChild . Person$
ISE04	Import a single object property with domain and range the same class $\top \sqsubseteq \forall hasChild \top . Person$ $\top \sqsubseteq \forall hasChild . Person$
ISE05	Import a single object property with domain multiple classes and range a class $\top \sqsubseteq \forall hasChild \top . Mother$ $\top \sqsubseteq \forall hasChild \top . Woman$ $\top \sqsubseteq \forall hasChild \top . Person$ $\top \sqsubseteq \forall hasChild . Person$
ISE06	Import a single object property with domain a class and range multiple classes $\top \sqsubseteq \forall hasChild \top . Person$ $\top \sqsubseteq \forall hasChild . Person$ $\top \sqsubseteq \forall hasChild . Human$ $\top \sqsubseteq \forall hasChild . Child$

(continued on next page)

(continued from previous page)

ISE07	Import a single datatype property with domain a class $\top \sqsubseteq \forall hasSSN^-.Person$
ISE08	Import a single datatype property with range rdfs:Literal $\top \sqsubseteq \forall hasName.rdfs:Literal$
ISE09	Import a single datatype property with domain a class and range rdfs:Literal $\top \sqsubseteq \forall hasName^-.Person$ $\top \sqsubseteq \forall hasName.rdfs:Literal$
ISE10	Import a single datatype property with domain multiple classes and range rdfs:Literal $\top \sqsubseteq \forall hasChildNamed^-.Mother$ $\top \sqsubseteq \forall hasChildNamed^-.Woman$ $\top \sqsubseteq \forall hasChildNamed.rdfs:Literal$

Group F: Property equivalences

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

Group G: Logical characteristics of properties

ISG01	Import a single transitive object property with domain and range the same class $hasFriend^+ \sqsubseteq hasFriend$ $\top \sqsubseteq \forall hasFriend^-.Person$ $\top \sqsubseteq \forall hasFriend.Person$

(continued on next page)

(continued from previous page)

ISG02	Import a single symmetric object property with domain and range the same class $hasFriend \equiv hasFriend^-$ $\top \sqsubseteq \forall hasFriend^-.Person$ $\top \sqsubseteq \forall hasFriend.Person$
ISG03	Import a single functional object property with domain a class and range another class $\top \sqsubseteq \leq 1 hasHusband$ $\top \sqsubseteq \forall hasHusband^-.Woman$ $\top \sqsubseteq \forall hasHusband.Man$
ISG04	Import a single functional datatype property with domain a class and range <code>rdfs:Literal</code> $\top \sqsubseteq \leq 1 hasAge$ $\top \sqsubseteq \forall hasAge^-.Person$ $\top \sqsubseteq \forall hasAge.rdfs:Literal$
ISG05	Import a single inverse functional object property with domain a class and range another class $\top \sqsubseteq \leq 1 hasTutor^-$ $\top \sqsubseteq \forall hasTutor^-.Professor$ $\top \sqsubseteq \forall hasTutor.Student$

Individual benchmarks

Group H: Single individuals

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

Group I: Named individuals and properties

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 $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$ $Person(MARY)$ $Person(PAUL)$ $hasChild(MARY, PAUL)$
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 $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$ $Person(PAUL)$ $knows(PAUL, PAUL)$
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 $\top \sqsubseteq \forall hasChild^-.Mother$ $\top \sqsubseteq \forall hasChild.Child$ $Mother(MARY)$ $Child(PAUL)$ $hasChild(MARY, PAUL)$
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 $\top \sqsubseteq \forall hasName^-.Person$ $\top \sqsubseteq \forall hasName.rdfs:Literal$ $Person(MARYSMITH)$ $hasName(MARYSMITH, "Mary")$
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 $\top \sqsubseteq \forall hasName^-.Person$ $\top \sqsubseteq \forall hasName.rdfs:Literal$ $Person(MARYANN)$ $hasName(MARYANN, "Mary")$ $hasName(MARYANN, "Ann")$

Group J: Anonymous individuals and properties

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 $\begin{aligned} \top &\sqsubseteq \forall hasChild^-.Person \\ \top &\sqsubseteq \forall hasChild.Person \\ &Person(JOHN) \\ &hasChild(ANON^a, JOHN) \end{aligned}$ ^aThis denotes an <i>anonymous individual</i>
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 datatype property with an individual of the other class $\begin{aligned} \top &\sqsubseteq \forall hasChild^-.Parent \\ \top &\sqsubseteq \forall hasChild.Person \\ &Person(JOHN) \\ &hasChild(ANON, JOHN) \end{aligned}$
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 $\begin{aligned} \top &\sqsubseteq \forall hasName^-.Person \\ \top &\sqsubseteq \forall hasName.rdfs:Literal \\ &hasName(ANON, "Peter") \end{aligned}$

Group K: Individual identity

ISK01	Import one class and two named individuals of the class that are the same $Person(MARYANN) = Person(MARY)$
ISK02	Import one class and two named individuals of the class that are different $\neg (Person(MARYANN) = Person(MARY))$
ISK03	Import one class and three named individuals of the class that are all of them different $\begin{aligned} &\neg (Person(MARY) = Person(ANN)) \\ &\neg (Person(MARY) = Person(JOAN)) \\ &\neg (Person(JOAN) = Person(ANN)) \end{aligned}$

Appendix E

The IBSE ontologies

This appendix describes the two ontologies used in the IBSE tool.

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 defines vocabulary for representing
    benchmarks.</rdfs:comment>
    <owl:versionInfo>24 October 2006</owl:versionInfo>
  </owl:Ontology>

  <!-- classes -->

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

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

  <!-- 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 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 defines vocabulary for representing
      benchmark results.</rdfs:comment>
    <owl:versionInfo>24 October 2006</owl:versionInfo>
  </owl:Ontology>

  <!-- classes -->

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

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

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

  <!-- 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>

```


Resumen amplio en español

En este anexo se incluye un resumen amplio de la tesis doctoral en español, tal y como requiere la Comisión de Doctorado de la Universidad Politécnica de Madrid.

En primer lugar se introduce el problema que se pretende resolver en esta tesis. A continuación se analiza el estado de la cuestión y se identifican los objetivos, que están basados en dicho análisis. Las secciones siguientes contienen un resumen de las propuestas presentadas en la tesis: una metodología de *benchmarking* de la tecnología de la Web Semántica, un marco de trabajo para la evaluación de la interoperabilidad de la tecnología de la Web Semántica (UPM-FBI), y la definición y ejecución de dos actividades internacionales de *benchmarking* que trataron la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio. Finalmente, se presentan las principales conclusiones y líneas futuras de trabajo en las áreas de *benchmarking* de la tecnología de la Web Semántica y del *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando un lenguaje de intercambio.

Introducción

La Web Semántica

La World Wide Web (también conocida como “WWW” o “Web”) es el universo de la información accesible en Internet, la encarnación del conocimiento humano¹. La Web está construida sobre un conjunto de software, protocolos y convenciones, que hacen posible que cualquiera recorra, navegue y contribuya a la misma, mediante el uso de técnicas de hipertexto y multimedia.

Aunque la información en la Web tiene la intención de ser útil y accesible tanto a los humanos como a las máquinas, la mayor parte de dicha información ha sido diseñada para el consumo humano y es difícil para los programas de ordenador manipularla de modo significativo y procesar su semántica. Así, la Web Semántica surge no como una Web separada sino como una extensión de la

¹<http://www.w3.org/WWW/>

Web actual, en la que se proporciona un significado bien definido a la información, permitiendo a los ordenadores y a las personas trabajar cooperativamente [Berners-Lee *et al.*, 2001]. Hoy en día, la Web Semántica es una red de datos e información que proporciona formatos comunes para representar conocimiento y reglas de inferencia, permitiendo agregar y combinar datos obtenidos de diferentes recursos.

Las tecnologías de la Web Semántica se encargan de manejar estos datos, ontologías (modelos de dominio) y reglas de inferencia. Aunque dichas tecnologías son altamente heterogéneas tanto en uso como en finalidad, normalmente se utilizan conjuntamente para realizar tareas en distintas fases del ciclo de vida de las ontologías, desde el desarrollo de las ontologías a su uso y mantenimiento, y para desarrollar aplicaciones semánticas.

Además, estas tecnologías necesitan un distinto grado de intervención humana, pueden ser utilizadas manualmente, semiautomáticamente o automáticamente; y proporcionan diferentes interfaces para acceder a ellas tales como interfaces de usuario, interfaces de programación, protocolos o servicios.

Evaluación de la tecnología de la Web Semántica

El uso a gran escala de la Web Semántica depende de dos tipos de evaluaciones, la evaluación de la tecnología que usa el contenido de la Web Semántica (evaluación de la tecnología de la Web Semántica) y la evaluación del contenido en si mismo (evaluación de las ontologías) [Sure *et al.*, 2004]. En esta tesis sólo tratamos con la evaluación de la tecnología.

La tecnología de la Web Semántica debe evaluarse como cualquier otro tipo de tecnología software. Hay numerosos métodos y herramientas para evaluar software tanto en la literatura propia del área de la Ingeniería Software [Sommerville, 2006] como en el área de la Experimentación de Software [Wohlin *et al.*, 2000, Boehm *et al.*, 2005], pero no son suficientes para evaluar la tecnología de la Web Semántica porque no cubren las características y usos específicos de dicha tecnología, tales como el uso de ontologías como modelos de datos, la suposición acerca de la incompletitud de la información del sistema (suposición de mundo abierto, conocida como “*open world assumption*”), la inferencia de nueva información o el uso de los estándares del W3C (RDF, OWL, etc.).

La evaluación de la tecnología de la Web Semántica comparte los mismos principios que cualquier otra evaluación de software, pero tiene distintos puntos de vista y objetivos. Esto no quiere decir que las evaluaciones de la tecnología de la Web Semántica deban realizarse desde cero. Las aproximaciones a la evaluación tradicionales de la Ingeniería Software pueden (y deben) ser utilizadas para evaluar la tecnología de la Web Semántica, pero además es necesario definir nuevos métodos de evaluación, infraestructuras y métricas para la tecnología de la Web Semántica, de forma que se puedan validar los resultados de investigación y se puedan mostrar los beneficios de dicha tecnología.

Además, los atributos de calidad utilizados para evaluar software pueden ser utilizados para evaluar la tecnología de la Web Semántica, pero teniendo

en cuenta las propiedades y el contexto de la Web Semántica. Por ejemplo, las preocupaciones en la Web Semántica acerca de la eficiencia no tienen que ver con la configuración del hardware o del software necesarios para ejecutar una aplicación sino con el tamaño de las ontologías que la aplicación maneja.

De todas formas, en el área de la Web Semántica la evaluación de la tecnología es escasa [Sure *et al.*, 2004], incluso teniendo en cuenta que en los últimos años distintos esfuerzos han aparecido en la comunidad en forma de: actividades de evaluación y *benchmarking*, tales como el entregable 1.3 [OntoWeb, 2002] de la Red Temática europea OntoWeb, la Iniciativa de Evaluación de Alineamiento de Ontologías² (*Ontology Alignment Evaluation Initiative*, OAEI), los *RDF(S)* y *OWL Interoperability Benchmarkings*³; talleres internacionales relacionados con la evaluación, tales como los talleres *Evaluation of Ontology-based Tools* (EON), el *Semantic Web Service Challenge*⁴ (*SWS Challenge*), los talleres *Scalable Semantic Web Knowledge Base Systems* (SSWS); y las conferencias *International y European Semantic Web Conference* (ISWC y ESWC, respectivamente), que suelen incluir sesiones relacionadas con la evaluación de la tecnología.

Además, hay un conjunto de casos de prueba que se han utilizado ampliamente en toda la comunidad de la Web Semántica: los *RDF Test Cases*⁵ [Grant and Beckett, 2004] y los *OWL Test Cases*⁶, y el *Lehigh University Benchmark* (LUBM) [Guo *et al.*, 2003, Guo *et al.*, 2005]).

En resumen, en los últimos años el número de actividades de evaluación y *benchmarking* está aumentando continuamente en el área de la Web Semántica. No obstante, la tecnología de la Web Semántica no ha sido suficientemente evaluada y el número de evaluaciones en esta área no es aún lo bastante bueno como para asegurar una tecnología de alta calidad.

Esto no es un comentario negativo, ya que la tecnología actual de la Web Semántica ha sido desarrollada principalmente en instituciones de investigación. Pero esto implica que las evaluaciones de tecnología son puntuales, se centran en validar resultados de investigación, no están suficientemente documentadas en los artículos de investigación donde normalmente se describen, son normalmente realizadas por una persona u organización, son ejecutadas bajo condiciones particulares, y tratan con un conjunto pequeño de herramientas de la Web Semántica (principalmente con herramientas de alineamiento de ontologías, herramientas de desarrollo de ontologías, repositorios de ontologías y razonadores) y no son aplicables, en general, a otros tipos de herramientas.

Esto hace que la evaluación de la tecnología de la Web Semántica sea difícil y cara, siendo esto una barrera importante para la transferencia de dicha tecnología al mercado, especialmente hoy en día cuando las empresas están reutilizando y desarrollando tecnología para la Web Semántica y están apareciendo empresas basadas en la Web Semántica.

Pero la gente no sabe cómo evaluar la tecnología de la Web Semántica, es

²<http://oaei.ontologymatching.org/>

³http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

⁴<http://sws-challenge.org/>

⁵<http://www.w3.org/TR/rdf-testcases/>

⁶<http://www.w3.org/TR/owl-test/>

difícil reutilizar resultados y lecciones aprendidas de otros, por lo que nuevos métodos y herramientas de evaluación tienen que ser desarrollados cuando la tecnología tiene que ser evaluada. Por otro lado, no hay ni métodos de evaluación estándar o consensuados ni herramientas que permitan la evaluación de la tecnología de la Web Semántica con respecto a un amplio rango de características (escalabilidad, interoperabilidad, usabilidad, etc.).

La necesidad de hacer *benchmarking* de la tecnología de la Web Semántica

Cualquier avance de investigación se basa en otros resultados de investigación existentes. En el caso de la investigación que acaba convirtiéndose en tecnología, avances de investigación simples requieren la reutilización y la mejora de desarrollos existentes. Por lo tanto, para que la investigación progrese es necesario que la tecnología sea evaluada, contrastada con otras y mejorada. Este argumento, válido para cualquier software, es también aplicable al software de la Web Semántica.

La idea de *benchmarking* como un proceso de búsqueda de mejora y de mejores prácticas (*best practices*) proviene de la concepción de *benchmarking* en la comunidad de la gestión empresarial [Camp, 1989, Spendolini, 1992]. Esta noción de *benchmarking* es similar a algunas aproximaciones en la Ingeniería Software [Wohlin *et al.*, 2002] pero difiere de otras en las que el *benchmarking* es considerado como un método de evaluación para la comparación de sistemas [Kitchenham, 1996, Weiss, 2002].

En esta tesis, definimos el *benchmarking* de software como un proceso colaborativo y continuo para mejorar los productos, servicios y procesos software mediante su evaluación sistemática y su comparación con aquellos considerados como los mejores [García-Castro, 2006c].

Aunque dentro del *benchmarking* se evalúa software, el *benchmarking* proporciona algunos beneficios que no se obtienen de las evaluaciones de software como una mejora continua del software, recomendaciones para desarrolladores sobre las prácticas utilizadas a la hora de desarrollar el software y, de esas prácticas, aquellas que pueden ser consideradas mejores prácticas.

El principal problema a la hora de hacer *benchmarking* de software es que no existe una metodología para ello. Además, las metodologías existentes para hacer *benchmarking* de procesos empresariales y otras metodologías de evaluación y mejora de software en la Ingeniería Software, tales como las de las áreas de Experimentación de Software o de Medición de Software, son generales y no están definidas en detalle siendo, por lo tanto, difíciles de utilizar en casos concretos.

El objetivo es obtener una mejora masiva de la tecnología actual de la Web Semántica proporcionando métodos y herramientas reutilizables, consensuados y libremente disponibles que puedan ser utilizados por diferentes personas en diferentes escenarios, válidos para los diferentes tipos de herramientas de la Web Semántica. Entonces, una evaluación y mejora continua de la tecnología de la Web Semántica sería posible mediante el *benchmarking* de dicha tecnología.

Esto requiere que se desarrollen métodos y herramientas genéricos, reutilizables, libremente disponibles y asequibles; se definan y se lleven a cabo las evaluaciones en consenso por diferentes grupos de personas en vez de por organizaciones individuales; y que las evaluaciones tengan lugar continuamente a lo largo del tiempo en vez de ser actividades puntuales.

La interoperabilidad de la tecnología de la Web Semántica

Esta tesis también trata de un problema importante en la Web Semántica, el de la interoperabilidad de la tecnología de la Web Semántica, y de la evaluación de dicha interoperabilidad.

Según el *Institute of Electrical and Electronics Engineers* (IEEE), la interoperabilidad es la habilidad que tienen dos o más sistemas o componentes de intercambiar información y de usar dicha información [IEEE-STD-610, 1991]. Duval propone una definición similar y declara que la interoperabilidad es la habilidad que tienen componentes software independientemente desarrollados de intercambiar información de forma que puedan ser utilizados conjuntamente [Duval, 2004]. Para nosotros, la interoperabilidad es la habilidad que tienen las herramientas de la Web Semántica de intercambiar ontologías y utilizarlas.

Uno de los factores que afecta a la interoperabilidad es la heterogeneidad. Sheth [1998] clasifica los niveles de heterogeneidad de cualquier sistema de información en heterogeneidad a nivel de información y heterogeneidad a nivel de sistema. En esta tesis, sólo se tiene en cuenta la heterogeneidad (y, por lo tanto, la interoperabilidad) a nivel de información.

Además, el problema de la interoperabilidad se trata en esta tesis en términos de reutilización de conocimiento y no debe confundirse con el problema de la interoperabilidad en términos de integración de recursos, estando este último relacionado con el problema de alineamiento de ontologías [Euzenat *et al.*, 2004a], es decir, el problema de encontrar correspondencias entre entidades en distintas ontologías.

Las ontologías permiten la interoperabilidad entre las distintas y heterogéneas tecnologías de la Web Semántica (herramientas de desarrollo de ontologías, repositorios de ontologías, herramientas de alineamiento de ontologías, razonadores, etc.). La interoperabilidad es una necesidad en las tecnologías de la Web Semántica porque dichos sistemas necesitan estar en comunicación para intercambiar ontologías y utilizarlas en el entorno distribuido y abierto de la Web Semántica. Por otro lado, la interoperabilidad es un problema en la Web Semántica debido a la heterogeneidad de los formalismos de representación de información de los distintos sistemas, ya que cada formalismo proporciona diferente expresividad de representación de conocimiento y diferentes capacidades de razonamiento, como ocurre en los sistemas basados en conocimiento [Brachmann and Levesque, 1985].

La mayoría de las herramientas manejan nativamente uno de los lenguajes recomendados por el W3C, ya sea RDF(S), OWL o ambos; pero existen herramientas que manejan modelos de representación diferentes a los de los lenguajes anteriores, siendo algunos de estos modelos más similares a los de RDF(S) y

OWL que otros. Ejemplos de estos lenguajes son el *Unified Modeling Language*⁷ (UML), el *Ontology Definition Metamodel*⁸ (ODM), o el lenguaje *Open Biomedical Ontologies*⁹ (OBO), cada uno con diferente expresividad a la hora de representar conocimiento y diferentes capacidades de razonamiento.

Idealmente, las ontologías que hayan sido definidas utilizando RDF(S) u OWL deberían ser intercambiadas correctamente entre las distintas herramientas que manejen estos lenguajes. No obstante, las herramientas actuales de la Web Semántica tienen problemas al intercambiar ontologías, ya provengan estas ontologías de otras herramientas o de la Web [Sure and Corcho, 2003, Corcho, 2005]. Algunas veces, los problemas surgen debido a los distintos formalismos de representación utilizados por las herramientas, ya que no todas ellas soportan nativamente RDF(S) u OWL; otras veces, los problemas son debidos a defectos en las herramientas.

Debido a esta heterogeneidad entre formalismos de representación en el escenario de la Web Semántica, el problema de la interoperabilidad está altamente relacionado con el problema de la traducción de ontologías que se da cuando ontologías comunes se comparten y reutilizan en múltiples sistemas de representación [Gruber, 1993].

Como hemos mencionado anteriormente, en la actualidad la Web Semántica está llena de herramientas que proporcionan funcionalidades limitadas y específicas. No ser consciente de la interoperabilidad entre las tecnologías de la Web Semántica existentes da lugar a problemas importantes a la hora de construir tecnologías y aplicaciones más complejas reutilizando las tecnologías existentes. Esta ignorancia con respecto a la interoperabilidad se debe principalmente a que la interoperabilidad entre las herramientas no es evaluada ya que no existe una manera fácil de realizar dichas evaluaciones.

Como previamente se vio en los talleres *Evaluation of Ontology-based Tools* (EON) [Sure and Corcho, 2003], la interoperabilidad entre las distintas herramientas de la Web Semántica no es sencilla. Descubrir por qué falla la interoperabilidad es difícil ya que cualquier suposición hecha para la traducción dentro de una herramienta puede fácilmente impedir una interoperabilidad exitosa con otras herramientas.

Estado de la Cuestión

Evaluación de software

La evaluación de software juega un papel importante en distintas áreas de la Ingeniería Software, tales como la Medición de Software (*Software Measurement*), la Ingeniería de Software Experimental (*Experimental Software Engineering*) o el Testeo de Software (*Software Testing*).

⁷<http://www.uml.org/>

⁸<http://www.omg.org/ontology/>

⁹<http://obofoundry.org/>

Según el estándar ISO 14598 [ISO/IEC, 1999], la evaluación de software es *la examinación sistemática de la capacidad de una entidad para cumplir con los requisitos especificados*, considerando software no sólo como un conjunto de programas de ordenador sino también como los procedimientos, documentación y datos producidos.

Las evaluaciones de software tienen lugar a lo largo de todo el ciclo de vida del software, pudiendo ser realizadas durante el proceso de desarrollo del software evaluando productos software intermedios o una vez se ha finalizado el desarrollo.

Aunque las evaluaciones se realizan normalmente dentro de la organización que desarrolla el software, otros grupos de personas tales como usuarios o auditores, independientes a la organización, pueden evaluar el software. El uso de terceros en evaluaciones de software puede ser muy efectivo, pero dichas evaluaciones son mucho más caras [Rakitin, 1997].

Los objetivos de evaluar software dependen de cada caso específico, pero pueden ser resumidos [Basili *et al.*, 1986, Park *et al.*, 1996, Gediga *et al.*, 2002] en los siguientes:

- *Describir* el software para entenderlo y establecer líneas base para comparaciones.
- *Evaluar* el software con respecto a ciertos requisitos o criterios de calidad y determinar el grado de calidad deseada del producto software y sus debilidades.
- *Mejorar* el software encontrando oportunidades para mejorar su calidad. Esta mejora se mide comparando el software con las líneas base.
- *Comparar* productos software alternativos o diferentes versiones de un mismo producto.
- *Controlar* la calidad del software asegurando que cumple con el nivel de calidad requerido.
- *Prever* para poder tomar decisiones, estableciendo nuevas metas y planes para cumplirlas.

El software puede ser medido de acuerdo a numerosos atributos de calidad. Múltiples modelos de calidad de software han sido definidos después de las primeras propuestas de Boehm [1976] y de Calvano y McCall [1978] en los años 70, tales como el marco de calidad de producto software descrito en el estándar ISO 9126 [ISO/IEC, 2001], uno de los más conocidos.

Benchmarking

En las últimas décadas, la palabra *benchmarking* ha sido relevante dentro de la comunidad de la gestión empresarial. Las definiciones más conocidas en esta área son las de Camp [1989] y Spendolini. Camp define *benchmarking* como la

búsqueda de las mejores prácticas de la industria que llevan a un rendimiento superior, mientras que Spendolini expande la definición de Camp añadiendo que *benchmarking* es un proceso continuo y sistemático para evaluar los productos, servicios y procesos de trabajo de organizaciones reconocidas como las que representan las mejores prácticas con el objetivo de la mejora organizativa. En este contexto, mejores prácticas son prácticas buenas que han funcionado bien en todos los lados, han sido probadas y han producido resultados exitosos [Wireman, 2003].

Estas definiciones destacan las dos características principales del *benchmarking*: la mejora continua y la búsqueda de mejores prácticas.

La comunidad de la Ingeniería Software también utiliza el término *benchmarking* aunque no comparte una definición común del mismo. A continuación presentamos algunas de las definiciones más representativas utilizadas en la comunidad de la Ingeniería Software:

- Kitchenham [1996] y Weiss [2002] definen *benchmarking* como un método de evaluación de software apropiado para comparaciones de sistemas. Para Kitchenham, *benchmarking* es el proceso de ejecutar una serie de pruebas estándar utilizando un número de herramientas/métodos alternativos y evaluar el rendimiento relativo de las herramientas en dichas pruebas, mientras que para Weiss *benchmarking* es un método para medir el rendimiento contra un estándar o un conjunto de estándares.
- Wohlin et al. [2002] adoptan la definición de *benchmarking* de la comunidad empresarial, considerando *benchmarking* como un proceso de mejora continua que procura ser el mejor de los mejores mediante la comparación de procesos similares en diferentes contextos.

El motivo de hacer *benchmarking* de productos software en vez de simplemente evaluarlos es el de obtener varios beneficios que no pueden conseguirse a través de evaluaciones de software. Una evaluación de software muestra las debilidades del software o su cumplimiento con los requisitos de calidad. Si varios productos software son evaluados, también se obtiene un análisis comparativo de dichos productos y recomendaciones para usuarios. Al hacer *benchmarking* de varios productos, además de todos los beneficios comentados, se obtiene una mejora continua sobre los productos, recomendaciones para desarrolladores sobre las prácticas utilizadas para desarrollar dichos productos y, de esas prácticas, aquellas que se pueden considerar mejores prácticas.

Metodologías de evaluación y mejora

Esta sección enumera las distintas metodologías que se han tenido en cuenta en el desarrollo de esta tesis relacionadas con la evaluación y mejora en las áreas de *benchmarking* en la gestión empresarial, Medición de Software y Experimentación de Software. Estas tres áreas proporcionan una visión general de distintos métodos que tratan con temas relevantes al *benchmarking* de software

como la evaluación como una actividad continua, una perspectiva a nivel de toda la empresa de la evaluación y la realización de experimentos sobre el software.

Las metodologías de *benchmarking* de la comunidad de la gestión empresarial ven el *benchmarking* como un mecanismo para mejorar los procesos de negocio de una empresa y lo consideran un proceso continuo. Dichas metodologías, al ser muy generales, pueden ser fácilmente adaptadas a la comunidad de la Ingeniería Software. Las metodologías mas relevantes de esta área son la propuesta por Camp [1989], la rueda de *benchmarking* propuesta por Andersen y Pettersen [1996], y la propuesta por el American Productivity and Quality Centre [Gee *et al.*, 2001].

Las metodologías de Medición de Software son utilizadas para implementar programas de medición de software en empresas. Tenemos en cuenta estas metodologías porque consideran la medición de software a lo largo de toda la empresa y también consideran la medición de software como un proceso continuo. Las metodologías mas relevantes de esta área son la propuesta por Grady y Caswell [1987], quienes describen la implementación de un programa de medición de software en Hewlett-Packard, el marco para desarrollar e implementar programas de medición de software en empresas de Goodman [1993], y el método propuesto por McAndrews [1993], que establece el proceso de medición como parte de un proceso general de software en una organización.

Las metodologías de Experimentación de Software son utilizadas para realizar experimentos sobre software. Al contrario que las aproximaciones anteriores, la experimentación de software se considera como una actividad puntual. No obstante, estas metodologías son relevantes ya que dentro del *benchmarking* se realizan experimentos sobre software. Las metodologías mas relevantes de esta área son el marco de experimentación propuesto por Basili *et al.* [1986], los seis pasos propuestos por Pfleeger [1995], el proceso general de experimentación propuesto por Wohlin *et al.* [2000] y las guías de Kitchenham *et al.* [2002] para realizar experimentos.

Esta lista de metodologías no es exhaustiva ni completa, ya que metodologías de otras áreas como la Evaluación de Software [ISO/IEC, 1999, Basili, 1985] o el Testeo de Software [Myers *et al.*, 2004], no consideradas en esta tesis, podrían haber sido tenidas en cuenta.

Evaluaciones anteriores de la interoperabilidad

En el área de la Web Semántica, la interoperabilidad de la tecnología ha sido evaluada puntualmente. Algunos análisis cualitativos fueron presentados en [OntoWeb, 2002] e incluían herramientas de desarrollo de ontologías, de mezcla e integración, de evaluación de ontologías, de anotación basadas en ontologías y de almacenamiento y consulta de ontologías; y en [Maynard *et al.*, 2007] donde se consideraban herramientas de anotación basadas en ontologías. Estos análisis proporcionan información sobre la interoperabilidad de las herramientas (tales como las plataformas en las que se pueden ejecutar, las herramientas con las que interoperan, o los formatos de datos y de ontologías que manejan), pero no proporcionan estudios empíricos que soporten sus conclusiones.

La única excepción son los experimentos del *Second International Workshop on Evaluation of Ontology-based Tools* (EON2003)¹⁰. El tópico central de este taller fue la evaluación de la interoperabilidad de las herramientas de desarrollo de ontologías utilizando un lenguaje de intercambio [Sure and Corcho, 2003].

En este taller, se pidió a los participantes que modelaran ontologías con sus herramientas de desarrollo de ontologías y que realizaran distintas pruebas para evaluar la importación, exportación e interoperabilidad de las herramientas.

Conclusiones

Mientras estuve buscando, sin éxito, una metodología de *benchmarking* de software al principio de mi tesis, me encontré con una serie de metodologías de evaluación y mejora.

Estas metodologías, que pertenecen a distintas áreas (gestión empresarial, Medición de Software y Experimentación de Software), pueden ser el punto de partida de una metodología de *benchmarking* de software ya que comparten ciertas formas de resolver sus problemas y poseen características relevantes al *benchmarking* de software.

No obstante, no se han considerado exhaustivamente todas las metodologías de evaluación y mejora existentes en dichas áreas, ya que son numerosas. Las metodologías que se han tenido en cuenta como punto de partida son aquellas que son bien conocidas y que proporcionan descripciones detalladas de sus procesos.

Con respecto al problema de la interoperabilidad de la tecnología de la Web Semántica, los experimentos del EON2003 fueron un primer paso valioso para evaluar la interoperabilidad, ya que destacaron los problemas de interoperabilidad en las herramientas existentes al utilizar los lenguajes recomendados por el W3C para intercambiar ontologías.

Aun así, son necesarias nuevas evaluaciones de la interoperabilidad de la tecnología de la Web Semántica porque:

- La interoperabilidad es un gran problema en la Web Semántica que está sin resolver en la actualidad.
- Los experimentos del taller sólo consideraban unas pocas herramientas y se centraban en herramientas de desarrollo de ontologías.
- Algunos experimentos evaluaban funcionalidades de exportación, otros de importación y sólo unos pocos la interoperabilidad. Además, la interoperabilidad entre una herramienta y ella misma utilizando un lenguaje de intercambio no fue considerada.
- No se realizó una evaluación sistemática; cada experimento utilizó distintos procedimientos de evaluación, lenguajes de intercambio y principios para modelar las ontologías. Por lo tanto, sólo se obtuvieron comentarios y recomendaciones específicas para cada herramienta que participó.

¹⁰<http://km.aifb.uni-karlsruhe.de/ws/eon2003/>

Planteamiento

Objetivos de la tesis y problemas abiertos

El objetivo de esta tesis es avanzar el estado de la cuestión actual en el área de la Web Semántica primero proporcionando **una metodología para hacer *benchmarking* de la tecnología de la Web Semántica** y, segundo, aplicando esta metodología en un ***benchmarking* de la interoperabilidad de la tecnología de la Web Semántica** utilizando RDF(S) y OWL como lenguajes de intercambio.

Para lograr el primer objetivo, la siguiente lista (no exhaustiva) de problemas de investigación abiertos debe ser resuelta:

- Desde una perspectiva metodológica, hay al menos tres problemas abiertos:
 - Aunque hay metodologías de *benchmarking* en la comunidad de la gestión empresarial, no hay metodología de *benchmarking* de software, siendo necesaria para obtener las mejores prácticas utilizadas en el desarrollo de la tecnología de la Web Semántica y para obtener una mejora continua en esta tecnología.
 - Las metodologías de *benchmarking* en el área de la gestión empresarial y las metodologías de evaluación y mejora de software en el área de la Ingeniería Software son generales y no están definidas en detalle, siendo difícil su utilización en casos concretos en el área de la Web Semántica.
 - No existen métodos y técnicas integrados que den soporte a la tarea compleja de hacer *benchmarking* de la tecnología de la Web Semántica. Varias aproximaciones específicas se han propuesto bajo la perspectiva del tipo de tecnología (herramientas de desarrollo de ontologías, herramientas de mezcla y alineamiento de ontologías, herramientas de anotación basadas en ontologías, razonadores, etc.), pero dichas aproximaciones son difíciles de reutilizar y mantener ya que son específicas para cada tipo de tecnología.
- Desde una perspectiva tecnológica, no existen herramientas que den soporte al *benchmarking* de distintos tipos de tecnologías de la Web Semántica y a las distintas tareas que se han de realizar en dichas actividades.

Con respecto al segundo objetivo de hacer un *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica, la siguiente lista (no exhaustiva) de problemas abiertos debe ser resuelta:

- Los límites de la interoperabilidad entre las herramientas actuales son desconocidos en el momento de empezar esta tesis.

- No hay ningún método que describa cómo hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando un lenguaje de intercambio.
- No hay conjuntos de pruebas disponibles que puedan ser reutilizados para evaluar y hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando un lenguaje de intercambio.
- No hay un soporte software específico para evaluar y hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando un lenguaje de intercambio.

Contribuciones al estado de la cuestión

Esta tesis trata de dar solución a los anteriores problemas de investigación abiertos.

Con respecto al primer objetivo, esta tesis avanza el estado de la cuestión en los siguientes aspectos:

1. **Desarrollo de una metodología de *benchmarking* de la tecnología de la Web Semántica**, basada en metodologías y prácticas existentes en otras áreas, tan general y abierta como sea posible de forma que pueda cubrir la amplia variedad de tecnologías de la Web Semántica. Esta metodología describe el proceso de *benchmarking* con su secuencia de tareas, actores, entradas y salidas. Esta metodología ha sido validada comprobando que cumple con las condiciones necesarias y suficientes que toda metodología debe tener y aplicándola en diferentes escenarios a distintos tipos de tecnologías de la Web Semántica con distintos criterios de evaluación.

El segundo objetivo de esta tesis es la aplicación de la metodología de *benchmarking* anterior para hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica. Esta tesis presenta avances en el estado de la cuestión en los siguientes aspectos:

2. **Método para hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio**. Este método ha sido definido instanciando la metodología de *benchmarking* de la tecnología de la Web Semántica mencionada anteriormente y proporciona un marco para comparar los resultados de interoperabilidad de los distintos tipos de herramientas de la Web Semántica. Este método es el utilizado en las dos actividades de *benchmarking* realizadas en esta tesis, es decir, el *RDF(S) Interoperability Benchmarking* y el *OWL Interoperability Benchmarking*.
3. **El UPM Framework for Benchmarking Interoperability**¹¹ (UPM-FBI) que incluye los recursos necesarios para hacer *benchmarking* de la

¹¹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio.

El UPM-FBI proporciona cuatro conjuntos de pruebas que contienen ontologías para ser utilizadas en evaluaciones de interoperabilidad y dos enfoques para realizar experimentos de interoperabilidad (uno manual y otro automático), conteniendo cada uno de ellos las distintas herramientas que dan soporte a la ejecución de los experimentos y al análisis de los resultados.

Concretamente, el UPM-FBI proporciona:

- 3.1. **Conjuntos de pruebas para evaluar y hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio.** Estos conjuntos de pruebas son los utilizados como entradas en el UPM-FBI. Tres conjuntos de pruebas han sido definidos para evaluar la interoperabilidad utilizando RDF(S) como lenguaje de intercambio, el *RDF(S) Import Benchmark Suite*, el *RDF(S) Export Benchmark Suite*, y el *RDF(S) Interoperability Benchmark Suite*; y un conjunto de pruebas ha sido definido para evaluar la interoperabilidad utilizando OWL como lenguaje de intercambio, el *OWL Lite Import Benchmark Suite*.
- 3.2. **Enfoques manual y automático para evaluar y hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio.** En el enfoque manual, los experimentos se realizan accediendo manualmente a las herramientas y, en el automático, la ejecución de los experimentos y el análisis de los resultados es automático.
- 3.3. **Herramientas software para evaluar y hacer *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio.** Estas herramientas software son las necesarias para ejecutar los experimentos en el UPM-FBI y han sido desarrolladas teniendo en cuenta su reusabilidad para que puedan ser utilizadas en otras evaluaciones. Dos herramientas dan soporte al enfoque manual, la herramienta *rdfsbs*, que ejecuta automáticamente parte de los experimentos, y la aplicación web *IRIBA*¹², que proporciona una forma fácil de analizar los resultados. Una herramienta da soporte al enfoque automático, la herramienta *IBSE*¹³, que ejecuta los experimentos y analiza los resultados automáticamente.

4. Un claro panorama de la interoperabilidad entre los distintos tipos de herramientas de la Web Semántica. Las actividades de

¹²<http://knowledgeweb.semanticweb.org/iriba/>

¹³http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

benchmarking de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio nos han proporcionado información detallada sobre la interoperabilidad actual de las herramientas que han participado en los mismos.

Metodología de *benchmarking* de la tecnología de la Web Semántica

Adoptando la idea de *benchmarking* de la comunidad de la gestión empresarial, definimos el *benchmarking* de software como un proceso colaborativo y continuo para mejorar los productos, servicios y procesos software mediante su evaluación sistemática y su comparación con aquellos que son considerados los mejores.

A partir de esta definición, derivamos el propósito de la metodología de *benchmarking* de la tecnología de la Web Semántica, el cual es proporcionar un proceso comprensible para realizar *benchmarking* de software de la Web Semántica, identificando la secuencia de tareas que componen el proceso, sus entradas y salidas y los actores que participan en dichas tareas.

Los escenarios de uso de esta metodología consideran los distintos tipos de tecnologías de la Web Semántica. Estas tecnologías pueden estar en fase de desarrollo o producción, siendo los escenarios principales de la metodología los siguientes:

- Mejorar un grupo de herramientas de la Web Semántica.
- Evaluar un grupo de herramientas de la Web Semántica.
- Mejorar una herramienta de la Web Semántica.
- Evaluar un proceso de *benchmarking*.

Actores del *benchmarking*

Las tareas del proceso de *benchmarking* son llevadas a cabo por distintos actores dependiendo de los papeles que hay que tomar en cada tarea. Los actores que participan en el proceso de *benchmarking* son:

- *Iniciador del benchmarking*. El iniciador del *benchmarking* es el miembro (o miembros) de una organización que realiza las primeras tareas del proceso de *benchmarking*. Su trabajo consiste en preparar una propuesta para realizar el *benchmarking* y en obtener la aprobación de la dirección de la organización para llevarlo a cabo.
- *Dirección de la organización*. La dirección de la organización tiene un papel primordial en el proceso de *benchmarking*, ya que tiene que aprobar su realización y dar soporte a los cambios que resulten del mismo. Además,

debe asignar recursos al *benchmarking* e integrar la planificación del *benchmarking* con la de la organización.

- *Equipo de benchmarking.* Una vez la dirección de la organización ha aprobado la propuesta de *benchmarking*, el equipo de *benchmarking* se constituye y será quien realice la mayoría de las tareas restantes del *benchmarking*.
- *Socios del benchmarking.* Los socios del *benchmarking* son las organizaciones que participan en el *benchmarking*. Todos los socios deben acordar los pasos a llevar a cabo y sus necesidades han de ser tenidas en cuenta.
- *Desarrolladores de la herramienta.* Los desarrolladores de la herramienta considerada en el *benchmarking* son los que implementarán los cambios necesarios en la misma para mejorarla teniendo en cuenta las recomendaciones del *benchmarking*. Algunos pueden también formar parte del equipo de *benchmarking* y, en este caso, se deben minimizar su parcialidad.

Proceso de *benchmarking*

El proceso de *benchmarking* definido en esta metodología es un proceso continuo que debería ser realizado indefinidamente para obtener una mejora continua en las herramientas que participan en el *benchmarking*.

El proceso de *benchmarking* está compuesto por una iteración del *benchmarking* que se repite constantemente. Cada iteración del *benchmarking* se compone de tres fases (*Planificación*, *Experimentación* y *Mejora*) y finaliza con una tarea de *Recalibrado*. Los objetivos principales de estas fases son:

- **Fase de Planificación.** Su objetivo principal es obtener un documento con la propuesta de *benchmarking* detallada. Este documento se usará como referencia a lo largo del *benchmarking*, y debe incluir toda la información relevante acerca del mismo: sus objetivos, beneficios y costes; el software (y sus funcionalidades) que será evaluado; las métricas que se utilizarán para evaluar dichas funcionalidades; y las personas involucradas en el *benchmarking*. Las últimas tareas de esta fase consisten en buscar otras organizaciones que quieran participar en el *benchmarking* con otro software con el fin de llegar a un acuerdo en la propuesta de *benchmarking* (tanto dentro de la organización como con las otras organizaciones) y en la planificación del *benchmarking*.
- **Fase de Experimentación.** En esta fase, las organizaciones deben definir y ejecutar los experimentos de evaluación para cada software que participe en el *benchmarking*. Los resultados de la evaluación deben ser recopilados y analizados, determinando las prácticas que conducen a los mismos e identificando qué prácticas son las mejores prácticas.
- **Fase de Mejora.** La primera tarea de esta fase comprende la escritura del informe sobre el *benchmarking*, el cual debe incluir: un resumen del proceso seguido, los resultados y las conclusiones de la experimentación,

recomendaciones para mejorar el software y las mejores prácticas encontradas en la experimentación. Los resultados del *benchmarking* deben ser comunicados a las organizaciones que participan en el mismo y finalmente, en varios ciclos de mejora, los desarrolladores del software deben realizar los cambios necesarios para mejorarlo y monitorizar esta mejora.

Mientras que las tres fases mencionadas anteriormente están dedicadas a la mejora de la herramienta, el objetivo de la tarea de *Recalibrado* es mejorar el proceso de *benchmarking* gracias a las lecciones aprendidas al realizar la iteración del *benchmarking*.

Organizando las actividades de *benchmarking*

El *RDF(S) Interoperability Benchmarking* y el *OWL Interoperability Benchmarking*, fueron organizados y llevados a cabo siguiendo la metodología de *benchmarking* de la tecnología de la Web Semántica previamente descrita, la cual proporciona una serie de guías generales que deben ser adaptadas a cada caso.

Nuestro objetivo es **evaluar y mejorar la interoperabilidad de la tecnología de la Web Semántica**, utilizando RDF(S) como lenguaje de intercambio en un caso y OWL en el otro.

En nuestro caso, para intercambiar ontologías las herramientas las almacenarán como ficheros de texto serializados utilizando la sintaxis RDF/XML, la sintaxis más utilizada por las herramientas de la Web Semántica.

En este escenario, la interoperabilidad depende de dos funcionalidades de las herramientas, aquella que lee una ontología almacenada en la herramienta y la escribe en un fichero RDF(S) u OWL (exportador de RDF(S)/OWL) y aquella que lee un fichero RDF(S) u OWL con una ontología y la almacena en la herramienta (importador de RDF(S)/OWL).

Toda la información relacionada con las actividades de *benchmarking* se recopiló en sendas propuestas de *benchmarking*. Con el fin de alcanzar una mayor audiencia, las propuestas de *benchmarking* fueron páginas web públicas¹⁴¹⁵.

En el primer *benchmarking*, el *RDF(S) Interoperability Benchmarking*, limitamos el alcance a un tipo de tecnología, las herramientas de desarrollo de ontologías. No obstante, el *benchmarking* era lo suficientemente general como para permitir la participación de otros tipos de herramientas. En el *OWL Interoperability Benchmarking*, se amplió la participación a cualquier tipo de tecnología de la Web Semántica.

Cualquier herramienta de la Web Semántica capaz de importar y exportar RDF(S) u OWL podía participar en el *RDF(S) Interoperability Benchmarking* o en el *OWL Interoperability Benchmarking*, respectivamente. En el caso del *RDF(S) Interoperability Benchmarking*, no sólo participaron herramientas de desarrollo de ontologías, sino también repositorios de RDF.

¹⁴http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

¹⁵http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/

Seis herramientas participaron en el *RDF(S) Interoperability Benchmarking*, tres herramientas de desarrollo de ontologías: KAON, Protégé (utilizando su implementación RDF) y WebODE; las otras tres fueron repositorios de RDF: Corese¹⁶, Jena¹⁷ y Sesame¹⁸. Los experimentos no siempre fueron realizados por los desarrolladores de las herramientas.

Nueve herramientas participaron en el *OWL Interoperability Benchmarking*: una herramienta de anotación basada en ontologías: GATE¹⁹; tres repositorios de ontologías: Jena²⁰, KAON2²¹ y SWI-Prolog²²; y cinco herramientas de desarrollo de ontologías: el NeOn toolkit²³, Protégé-Frames²⁴, Protégé-OWL²⁵, Sementalk²⁶ y WebODE²⁷.

Cualquier grupo de ontologías podría utilizarse como entrada a los experimentos, pero utilizar ontologías reales, grandes o complejas puede ser inútil si no sabemos si las herramientas pueden intercambiar correctamente ontologías simples. Ya que uno de los objetivos del *benchmarking* es la mejora de las herramientas, las ontologías deben ser simples para permitir identificar posibles problemas y aislar sus causas.

Por lo tanto, como se verá en las siguientes secciones, el autor de esta tesis definió cuatro conjuntos de pruebas para utilizar en las actividades de *benchmarking*, los cuales fueron comunes para todas las herramientas.

La calidad de estos conjuntos de pruebas es esencial para obtener buenos resultados en el *benchmarking*. Así, una vez los conjuntos de pruebas fueron definidos, fueron publicados en páginas web para que fueran revisados por los participantes. Además, los conjuntos de pruebas fueron revisados en reuniones de Knowledge Web.

RDF(S) Interoperability Benchmarking

Para obtener los datos necesarios de los experimentos del *RDF(S) Interoperability Benchmarking*, el autor de la tesis definió los siguientes tres conjuntos de pruebas para evaluar la importación, exportación e interoperabilidad de las herramientas:

- El *RDF(S) Import Benchmark Suite* sirve para evaluar las funcionalidades de importación de RDF(S) de las herramientas de la Web Semántica. Este conjunto de pruebas puede ser utilizado en cualquier herramienta

¹⁶<http://www-sop.inria.fr/acacia/soft/corese/>

¹⁷<http://jena.sourceforge.net/>

¹⁸<http://www.openrdf.org/>

¹⁹<http://gate.ac.uk/>

²⁰<http://jena.sourceforge.net/>

²¹<http://kaon2.semanticweb.org/>

²²<http://www.swi-prolog.org/packages/semweb.html>

²³<http://www.neon-toolkit.org/>

²⁴<http://protege.stanford.edu/>

²⁵<http://protege.stanford.edu/overview/protege-owl.html>

²⁶<http://www.semtalk.com/>

²⁷<http://webode.dia.fi.upm.es/>

capaz de importar RDF(S). Cada prueba que lo compone define una ontología RDF(S) serializada en RDF/XML que debe ser importada por la herramienta.

El conjunto de pruebas distingue entre pruebas que comprueban la importación de distintas combinaciones de componentes del modelo de conocimiento de RDF(S) y pruebas que comprueban la importación de las distintas variantes de la sintaxis RDF/XML.

- El *RDF(S) Export Benchmark Suite* sirve para evaluar las funcionalidades de exportación a RDF(S) de las herramientas de la Web Semántica. Este conjunto de pruebas puede ser utilizado en cualquier herramienta capaz de exportar a RDF(S). Cada prueba que lo compone define una ontología que debe ser modelada en la herramienta y exportada a un fichero RDF(S).

El conjunto de pruebas distingue entre pruebas que comprueban la exportación de distintas combinaciones de componentes de los modelos de conocimiento de las herramientas y pruebas que comprueban la exportación de las distintas variantes de nombrado de componentes que permiten las herramientas y que están restringidas por RDF(S).

- El *RDF(S) Interoperability Benchmark Suite* sirve para evaluar la interoperabilidad de las herramientas de la Web Semántica utilizando RDF(S) como lenguaje de intercambio. Cada prueba que lo compone define una ontología que debe ser modelada en una herramienta origen, exportada a un fichero RDF(S) e importada por una herramienta destino.

Las ontologías del *RDF(S) Interoperability Benchmark Suite* son idénticas a las definidas para el *RDF(S) Export Benchmark Suite*, ya que en ambas se considera el subconjunto común de los modelos de conocimiento de KAON, Protégé, WebODE y RDF(S): clases y jerarquías de clases, propiedades de objeto y de tipo de datos, instancias y literales.

Resultados de importación de RDF(S)

En el *benchmarking*, las únicas herramientas nativas RDF(S) fueron Corese, Jena y Sesame, los repositorios de RDF. Las herramientas de desarrollo de ontologías (KAON, Protégé y WebODE) tienen sus propios modelos de conocimiento en los que algunos componentes pueden representarse en RDF(S) mientras que otros no.

Todas las herramientas que no tienen RDF(S) como modelo de conocimiento nativo tienen problemas de ejecución, tanto con componentes que son capaces de modelar como con aquellos que no. Por el contrario, las herramientas que tienen RDF(S) como modelo de conocimiento ejecutan correctamente todas las pruebas.

Por lo tanto, los resultados obtenidos al importar de RDF(S) dependen principalmente del modelo de conocimiento de la herramienta. Las herramientas que soportan RDF(S) nativamente, los repositorios RDF, no necesitan realizar traducciones al importar las ontologías e importan correctamente todas las

combinaciones de componentes. Las herramientas con modelos de conocimiento distintos a RDF(S), tienen que traducir las ontologías de RDF(S) a sus modelos.

En general, las herramientas de desarrollo de ontologías importan correctamente desde RDF(S) la mayoría de las combinaciones de componentes que modelan, raramente añadiendo o perdiendo información. En particular:

- KAON importa correctamente todas las combinaciones de componentes que puede modelar.
- Protégé sólo tiene problemas al importar clases o instancias que son instancias de múltiples clases.
- WebODE sólo tiene problemas al importar propiedades con rango un tipo de datos XML Schema.

Cuando las herramientas de desarrollo de ontologías importan ontologías con combinaciones de componentes que no pueden modelar, pierden la información acerca de dichos componentes. No obstante, suelen tratar de representar parcialmente esos componentes utilizando otros componentes de sus modelos de conocimiento. En la mayoría de los casos la importación se realiza correctamente. Las únicas excepciones son:

- KAON tiene problemas al importar jerarquías de clases con ciclos.
- Protégé tiene problemas al importar jerarquías de clases y propiedades con ciclos y propiedades con múltiples dominios.
- WebODE tiene problemas al importar propiedades con múltiples dominios o rangos.

Cuando las herramientas de desarrollo de ontologías manejan las distintas variantes de la sintaxis RDF/XML:

- Importan correctamente recursos con las distintas sintaxis de referencias URI.
- Importan correctamente recursos con las diferentes sintaxis (abreviadas y no abreviadas) de nodos vacíos, múltiples propiedades, nodos tipados, literales que son cadenas de caracteres y nodos en blanco (*blank nodes*). Las únicas excepciones son KAON cuando importa recursos con múltiples propiedades en la sintaxis no abreviadas y Protégé cuando importa recursos con nodos vacíos y en blanco en la sintaxis no abreviadas.
- No importan atributos de identificación de lenguaje (*xml:lang*) en etiquetas.

Resultados de exportación a RDF(S)

Sólo KAON y Protégé tienen algunos problemas al exportar ontologías a RDF(S). El resto de herramientas exportan correctamente las ontologías a RDF(S).

Al igual que con los resultados de importación, los resultados de exportación dependen del modelo de conocimiento de la herramienta. Las herramientas que soportan nativamente el modelo de conocimiento de RDF(S) (Corese, Jena y Sesame) no tienen que traducir al exportar las ontologías, mientras que las herramientas que no soportan nativamente RDF (KAON, Protégé y WebODE) tienen que traducir.

Por lo general, las herramientas de desarrollo de ontologías exportan correctamente la mayoría de las combinaciones de componentes que modelas sin perder información. En particular:

- KAON tiene problemas al exportar a RDF(S) propiedades de tipo de datos sin rango y propiedades de tipo de datos con múltiples dominios y rango un tipo de datos XML Schema.
- Protégé tiene problemas al exportar a RDF(S) clases o instancias que son instancias de múltiples clases y *template slots* con múltiples dominios.
- WebODE exporta correctamente a RDF(S) todas las combinaciones de componentes.

Cuando las herramientas de desarrollo de ontologías exportan componentes que están presentes en sus modelos de conocimiento pero que no pueden ser representados en RDF(S), tales como sus propios tipos de datos, normalmente introducen nueva información en la ontología aunque también pierden información.

Cuando manejan conceptos y propiedades cuyos nombres no cumplen las restricciones de caracteres de URI, cada herramienta de desarrollo de ontologías se comporta de forma distinta:

- Cuando los nombres no comienzan con una letra o con “_”, algunas herramientas no cambian el nombre y otras reemplazan el primer carácter con “_”.
- Los espacios en nombres son reemplazados por “-” o por “_”, dependiendo de la herramienta.
- Los caracteres reservados URI y los caracteres delimitadores de XML o no se cambian, o se reemplazan por “_”, o se codifican, dependiendo de la herramienta.

Resultados de interoperabilidad utilizando RDF(S)

Los resultados de importación y exportación mostraban pocos problemas al importar y exportar ontologías. Aun sí, las herramientas de desarrollo de

ontologías tienen problemas de interoperabilidad, tanto con los componentes que son capaces de modelar como con los que no.

Como comentario general, la interoperabilidad depende de:

- a. El funcionamiento correcto de los importadores y exportadores a RDF(S).
- b. La forma elegida para serializar las ontologías exportadas en la sintaxis RDF/XML.

Además, hemos observado que algunos problemas en cualquiera de esos factores afectan los resultados no de una sino de varias pruebas. Esto significa que en algunos casos corrigiendo un único problema o cambiando la forma de serializar las ontologías se pueden lograr importantes mejoras en la interoperabilidad.

Por otro lado, la interoperabilidad es baja cuando las herramientas intercambian ontologías que contienen restricciones de caracteres URI en nombres de clases y propiedades. Esto es debido principalmente a que las herramientas codifican algunos o todos los caracteres que no cumplen con dichas restricciones, lo cual provoca cambios en los nombres de clases y propiedades.

Interoperabilidad utilizando la misma herramienta

Las herramientas no tienen problemas al intercambiar consigo mismas ontologías que contienen componentes del subconjunto de sus modelos de conocimiento tenido en cuenta en las pruebas. La única excepción está en Protégé al intercambiar recursos que son instancia de múltiples clases, ya que no importa dichos recursos.

Interoperabilidad entre cada par de herramientas

La interoperabilidad entre distintas herramientas varía según las herramientas de que se trate. Además, en algunos casos las herramientas son capaces de intercambiar componentes en una dirección pero no en la otra.

Cuando **KAON** interopera con **Protégé**, pueden intercambiar correctamente algunos de los componentes que las herramientas modelan, pero tienen problemas con clases que son instancia de una metaclase o de múltiples metaclases, con propiedades de tipo de datos sin dominio o rango, con propiedades de tipo de datos cuyo rango es *String*, con instancias de múltiples clases y con instancias relacionadas mediante propiedades de tipo de datos.

Cuando **KAON** interopera con **WebODE**, pueden intercambiar correctamente casi todos los componentes que las herramientas modelan. La única excepción ocurre al intercambiar propiedades de tipo de datos con dominio y con rango *String*.

Cuando **Protégé** interopera con **WebODE**, pueden intercambiar correctamente todos los componentes que las herramientas modelan.

Interoperabilidad entre todas las herramientas

La interoperabilidad entre **KAON**, **Protégé** y **WebODE** se puede lograr mediante casi todos los componentes comunes que dichas herramientas modelan: clases, jerarquías de clases sin ciclos, propiedades de objeto con un dominio y un rango, instancias de una única clase e instancias relacionadas mediante propiedades objeto. Los únicos componentes comunes que estas herramientas no pueden utilizar son propiedades de tipo de datos con dominio y con rango *String* e instancias relacionadas mediante dichas propiedades.

OWL Interoperability Benchmarking

Para realizar los experimentos en el *OWL Interoperability Benchmarking* necesitamos una forma uniforme y automática de acceder a las herramientas de la Web Semántica que sea soportada por la mayoría de dichas herramientas. Debido a la alta heterogeneidad de las mismas, las APIs de manejo de ontologías varían mucho de una herramienta a otra. Por lo tanto, la forma elegida para acceder automáticamente a las herramientas es a través de las siguientes dos operaciones que soportan la mayoría de las herramientas de la Web Semántica: importar una ontología de un fichero y exportar una ontología a un fichero.

Durante el experimento, un grupo común de pruebas se ejecuta y cada prueba describe una ontología OWL que debe ser intercambiada entre una herramienta y las demás (incluida ella misma).

Las ontologías utilizadas como entrada a los experimentos son las del *OWL Lite Import Benchmark Suite*, descrita en detalle en [David *et al.*, 2006]. Este conjunto de pruebas se definió con el fin de evaluar la importación de OWL de las herramientas de la Web Semántica comprobando la importación de ontologías con combinaciones simples de componentes del modelo de conocimiento de OWL Lite. Está compuesta por 82 pruebas y está disponible en la Web²⁸.

Como la sintaxis RDF/XML permite serializar ontologías de distintas maneras manteniendo su semántica, el conjunto de pruebas incluye dos tipos de pruebas: unas para comprobar la importación de las distintas combinaciones de componentes de los términos de OWL Lite y otra para comprobar la importación de las distintas variantes de la sintaxis RDF/XML.

Para realizar los experimentos automáticamente y generar visualizaciones de los resultados, se desarrolló la herramienta IBSE²⁹ (*Interoperability Benchmark Suite Executor*). La herramienta IBSE ha sido implementada utilizando Java y su código fuente y binarios están disponibles en su página web³⁰.

Esta herramienta utiliza las ontologías *benchmarkOntology* y *resultOntology* para representar las pruebas y sus resultados y permite insertar nuevas herramientas para realizar experimentos.

²⁸http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html

²⁹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

³⁰http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/

La ejecución de IBSE comprende los siguientes pasos consecutivos, aunque también pueden ser ejecutados independientemente. Estos pasos son los siguientes:

1. **Generar a partir de un grupo de ontologías descripciones de las pruebas que puedan ser procesadas por un ordenador.** En este paso, a partir de un grupo de ontologías en una URI, se genera un fichero RDF con una prueba por cada ontología, utilizando el vocabulario de la ontología *benchmarkOntology*. La generación de estas descripciones se puede saltar si estas ya se encuentran disponibles.
2. **Ejecutar las pruebas.** En este paso, teniendo en cuenta todas las posibles combinaciones de intercambio de ontologías entre las herramientas, cada prueba descrita en el fichero RDF se ejecuta y sus resultados se almacenan en otro fichero RDF, utilizando el vocabulario de la ontología *resultOntology*.

Para ejecutar una prueba entre una herramienta origen y otra destino, primero el fichero con la ontología se importa en la herramienta origen y se exporta a un fichero intermedio y, segundo, este fichero intermedio se importa en la herramienta destino y se exporta a un fichero final.

Una vez se tienen los ficheros inicial, intermedio y final con sus correspondientes ontologías, se extraen los resultados de la ejecución comparando las ontologías entre ellas. Esta comparación y su salida dependen de un comparador de ontologías externo. La implementación actual utiliza el comparador de OWL de las *KAON2 OWL Tools*³¹, aunque otros comparadores se pueden introducir implementando un interfaz Java.

3. **Generar ficheros HTML con diferentes visualizaciones de los resultados.** En este paso, se generan distintos ficheros HTML con diferentes visualizaciones, resúmenes y estadísticas de los resultados.

Resultados de interoperabilidad utilizando OWL

Una vez que la herramienta IBSE ha sido adaptada para incluir todas las herramientas que participan en el *benchmarking*, se llevaron a cabo los experimentos. Se obtuvieron resultados de interoperabilidad para ocho herramientas: GATE, Jena, KAON2, Protégé-Frames, Protégé-OWL, SemTalk, SWI-Prolog, y WebODE. El autor de esta tesis ejecutó los experimentos, recopiló los resultados de la ejecución, puso disponibles dichos resultados (tanto los ficheros HTML como los RDF) en la página web del *benchmarking*³² y realizó un análisis detallado de los mismos, incluyendo resultados concretos para cada herramienta.

Lo primero que se puede comprobar a simple vista es que la interoperabilidad entre las herramientas es baja, incluso en intercambios entre una herramienta y ella misma.

³¹versión 0.27 <http://owltools.ontoware.org/>

³²http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/2007-08-12_Results/

También queda claro, como estaba previsto, que la interoperabilidad utilizando OWL como lenguaje de intercambio depende del modelo de conocimiento de las herramientas, cuanto el modelo de conocimiento de una herramienta es más similar al de OWL, la herramienta es más interoperable. No obstante, la forma de serializar las ontologías en la sintaxis RDF/XML también influye altamente en los resultados.

Que los importadores y exportadores de las herramientas funcionen correctamente no asegura la interoperabilidad de las mismas. Los intercambios entre Jena y Protégé OWL y los intercambios entre Jena y SWI-Prolog producen las mismas ontologías. Pero en los intercambios entre Protégé OWL y SWI-Prolog, cuando el intercambio va de Protégé OWL a SWI-Prolog hay algunos problemas³³. Esto nos lleva a un segundo hecho, la interoperabilidad entre dos herramientas es normalmente diferente dependiendo de la dirección del intercambio.

Jena, KAON2, Protégé OWL y SWI-Prolog pueden intercambiar correctamente todas las combinaciones de componentes excepto jerarquías de clases, equivalencias de clases y jerarquías de propiedades. Jena, Protégé OWL y SWI-Prolog pueden intercambiar correctamente todas las combinaciones de componentes pero, debido al problema mencionado anteriormente entre Protégé OWL y SWI-Prolog, los únicos conjuntos de herramientas totalmente interoperables son Jena con Protégé OWL y Jena con SWI-Prolog.

Con respecto a la robustez de las herramientas, éstas no tienen problemas al manejar las ontologías del conjunto de pruebas pero algunas de ellas tienen problemas de ejecución al procesar ontologías generadas por otras herramientas. Por supuesto, esta falta de robustez tiene un efecto negativo en la interoperabilidad.

Conclusiones

El objetivo del trabajo presentado en esta tesis ha sido avanzar el estado de la cuestión sobre *benchmarking* de la tecnología de la Web Semántica con dos objetivos principales:

- El desarrollo de una metodología de *benchmarking* de la tecnología de la Web Semántica, basada en metodologías y prácticas existentes en otras áreas, tan general y abierta como sea posible de forma que pueda ser utilizada en un amplio rango de tecnologías de la Web Semántica.
- La aplicación de esta metodología de *benchmarking* realizando un *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio. Lo cual incluye el proporcionar métodos específicos, tecnología y conjuntos de pruebas que den soporte a este *benchmarking*.

³³SWI-Prolog produce ontologías con un identificador de *namespace* incorrecto (`[/]`) al importar ontologías con *namespaces* por defecto (`xmlns="namespaceURI"`)

A continuación se presentan las conclusiones relacionadas con las principales contribuciones de esta tesis al estado de la cuestión.

Desarrollo y uso de la metodología de *benchmarking*

Realizar actividades de *benchmarking* sobre las tecnologías de la Web Semántica es una tarea difícil debido a que no existe una metodología de *benchmarking* de software y a que las metodologías de evaluación y mejora actuales son difíciles de utilizar con las tecnologías de la Web Semántica porque no están definidas en detalle.

Una de las contribuciones de esta tesis es el desarrollo de una metodología de *benchmarking* de la tecnología de la Web Semántica. Esta metodología de *benchmarking* es colaborativa y abierta, y ha sido definida partiendo de conocidas metodologías de evaluación y *benchmarking* en otras áreas, reutilizando tareas comunes de dichas metodologías.

La metodología propuesta en esta tesis no ha sido formalmente evaluada. No obstante, ha sido validada comprobando que cumple con las condiciones formales y materiales de adecuación de toda metodología [Paradela, 2001] y su aplicabilidad ha sido probada al utilizarla en las actividades de *benchmarking* realizadas en la Red de Excelencia Knowledge Web donde esta metodología ha sido utilizada para hacer *benchmarking* de herramientas de desarrollo de ontologías (tal y como se ve en esta tesis), de herramientas de alineamiento de ontologías [Euzenat *et al.*, 2004b], y de razonadores [Huang *et al.*, 2007]. Una aproximación para hacer *benchmarking* del rendimiento y la escalabilidad de herramientas de desarrollo de ontologías utilizando esta metodología se encuentra en [García-Castro and Gómez-Pérez, 2005a].

No podemos asegurar que la metodología sea válida en cualquier escenario, pero la metodología será validada en el futuro en nuevas actividades de *benchmarking* en distintas condiciones.

En esta tesis hemos probado que es viable evaluar y mejorar distintos tipos de tecnologías de la Web Semántica mediante un método común, siguiendo un enfoque basado en el problema en vez de en las herramientas. Este es el caso de las dos aplicaciones de la metodología mostradas en esta tesis, donde se ha realizado el *benchmarking* de la interoperabilidad de la tecnología de la Web Semántica utilizando RDF(S) y OWL como lenguajes de intercambio. No obstante, la metodología ha sido también utilizada en otras actividades de *benchmarking* sobre herramientas de desarrollo de ontologías, herramientas de alineamiento de ontologías y razonadores.

Después de utilizar la metodología en los dos casos de estudio, podemos dar algunas recomendaciones para realizar *benchmarking*:

- La participación de expertos reconocidos en la comunidad a lo largo de todo el proceso de *benchmarking* es crucial, y la inclusión de las mejores herramientas es una necesidad, incluso en los casos en los que las organizaciones que desarrollan dichas herramientas no participen en el *benchmarking*.

- El *benchmarking* requiere tiempo, ya que hay que realizar tareas que no son inmediatas como anuncios, acuerdos, etc. Por lo tanto, debe empezar pronto, su planificación debe considerar una duración realista del *benchmarking* y se le deben asignar suficientes recursos.
- Hemos observado que el esfuerzo que requiere el *benchmarking* es un criterio principal para que una organización decida si participar o no, especialmente para las empresas. Los recursos son necesarios principalmente en tres tareas: la organización del *benchmarking*, la definición de los experimentos, y la ejecución de los experimentos y el análisis de los resultados de los mismos. Por lo tanto, las tareas del *benchmarking*, particularmente las relacionadas con la experimentación, deberían realizarse automáticamente tanto como sea posible.

***Benchmarking* de la interoperabilidad**

Las actividades de evaluación y *benchmarking* actuales sobre la tecnología de la Web Semántica son escasas y un impedimento para el desarrollo total y la madurez de dicha tecnología. La Web Semántica necesita producir métodos y herramientas para la evaluación de la tecnología a gran escala de una forma fácil y económica. Para esto es necesario definir evaluaciones de la tecnología teniendo en cuenta su reusabilidad.

Al inicio de esta tesis, no existían métodos, conjuntos de pruebas y herramientas que pudieran ser reutilizados para realizar evaluaciones y actividades de *benchmarking* sobre las tecnologías de la Web Semántica utilizando un lenguaje de intercambio.

Esta tesis contribuye al estado de la cuestión definiendo el *UPM Framework for Benchmarking Interoperability* (UPM-FBI). Este marco de trabajo comprende dos enfoques para hacer *benchmarking* de la interoperabilidad utilizando un lenguaje de intercambio, uno manual y otro automático, y proporciona herramientas y conjuntos de pruebas que dan soporte a dichos enfoques.

Los cuatro conjuntos de pruebas definidos en esta tesis satisfacen las ocho propiedades deseables identificadas para conjuntos de pruebas. Como estos conjuntos de pruebas están disponibles públicamente, pueden ser utilizados tanto por desarrolladores de herramientas para evaluar y mejorar sus herramientas o por ingenieros ontológicos para seleccionar la herramienta apropiada para sus actividades de desarrollo de ontologías.

No obstante, hay que destacar que los conjuntos de pruebas presentados en esta tesis han sido definidos con el objetivo de evaluar la interoperabilidad. Por lo tanto, aunque estos conjuntos de pruebas pueden ser utilizados para evaluar los importadores y exportadores de las herramientas, no son completos para dichas tareas. Una evaluación exhaustiva de los importadores de RDF(S)/OWL debería tener en cuenta todo el modelo de conocimiento de RDF(S)/OWL y una evaluación exhaustiva de los exportadores debería tener en cuenta todo el modelo de conocimiento de la herramienta.

Con respecto a los dos enfoques del UPM-FBI, el enfoque manual tiene ciertas ventajas sobre el automático y viceversa. El esfuerzo necesario para ejecutar los experimentos y la calidad de los resultados depende de la involucración de personas y, mientras la experimentación automática es más barata, flexible y extensible, la calidad de los resultados producidos por personas es mayor.

El enfoque a seguir dependerá de las necesidades específicas del *benchmarking* pero, en general, el enfoque automático es recomendado porque un menor coste en el *benchmarking* y un mayor número de participantes es preferible al incremento en la calidad de los resultados que el enfoque manual proporciona.

También hemos desarrollado la herramienta IBSE, una herramienta fácil de utilizar para la evaluación a gran escala de la interoperabilidad de la tecnología de la Web Semántica utilizando un lenguaje de intercambio.

La herramienta IBSE puede ser utilizada en otros escenarios, utilizando cualquier grupo de ontologías como entrada o utilizando otros lenguajes de intercambio. Actualmente, la herramienta permite realizar los experimentos utilizando RDF(S) como lenguaje de intercambio y *rdf-utils*³⁴ como comparador de ontologías.

Resultados de la interoperabilidad utilizando RDF(S) y OWL

Al comienzo de esta tesis se desconocían los límites de la interoperabilidad entre las herramientas de la Web Semántica.

La contribución principal con respecto a este tema es la evaluación de la interoperabilidad actual entre varias herramientas de la Web Semántica bien conocidas: seis en el *RDF(S) Interoperability Benchmarking* y nueve en el *OWL Interoperability Benchmarking*. Esta evaluación nos ha proporcionado resultados detallados acerca del comportamiento de las herramientas no sólo cuando interoperan con otras herramientas sino también cuando importan y exportan ontologías RDF(S) y OWL.

Los resultados del *benchmarking* están disponibles públicamente en la Web de forma que cualquiera puede utilizarlos. No obstante, hay que destacar que dichos resultados son válidos para las versiones específicas de las herramientas en las que se ejecutaron los experimentos y, ya que el desarrollo de las herramientas continúa, se espera que dichos resultados cambien. Esto destaca la necesidad de la evaluación continua de la tecnología.

La conclusión más importante que se obtiene de los resultados es que la interoperabilidad entre las herramientas es baja y los conjuntos de herramientas interoperables son mínimos. Además, la interoperabilidad utilizando un lenguaje de intercambio depende altamente en los modelos de conocimiento de las herramientas. La interoperabilidad es mejor cuando el modelo de conocimiento de las herramientas es más similar al del lenguaje de intercambio. En los casos en los que los modelos de conocimiento difieren, la interoperabilidad sólo se puede lograr utilizando ontologías ligeras (*lightweight*).

³⁴versión 0.3b <http://wymiwyg.org/rdf-utils/>

Esta panorámica, aunque desalentadora, puede servir para promover el segundo de nuestros objetivos, la mejora de las herramientas. Aunque este objetivo está actualmente fuera de nuestro alcance, ya que cada herramienta es desarrollada en organizaciones independientes, hemos mostrado cómo hemos logrado esta mejora en WebODE y esperamos, no obstante, que los resultados que hemos obtenido puedan ayudar en la mejora de las herramientas.

Una interoperabilidad real en la Web Semántica requiere la involucración de los desarrolladores de las herramientas. Los desarrolladores de las herramientas que han participado en las actividades de *benchmarking* han sido informados acerca de los resultados de sus herramientas en dichas actividades y de las recomendaciones propuestas para mejorar sus herramientas.

No obstante, algunas de las herramientas que han participado han sido mejoradas incluso antes de la fase de *Mejora* de la metodología. Como nuestro objetivo era la mejora, se permitieron modificaciones en las herramientas en cualquier momento y, en algunos casos, los desarrolladores mejoraron sus herramientas al ejecutar los experimentos.

Después de analizar los resultados, hemos comprobado que el problema de la interoperabilidad no sólo depende del problema de la traducción de ontologías sino también de problemas de robustez y especificación, es decir, de decisiones de desarrollo tomadas por los desarrolladores.

Aparte de este comentario general, no podemos decir que hay un grupo de problemas de interoperabilidad “típicos”, ya que los resultados de interoperabilidad dependen en gran parte de las herramientas que participan en el intercambio y los comportamientos de cada herramienta son diferentes.

Futuras líneas

Aunque esta tesis presenta contribuciones importantes al estado de la cuestión en *benchmarking* de la tecnología de la Web Semántica, todavía hay algunos problemas de investigación abiertos que no se han tenido en cuenta en esta tesis o que han surgido como consecuencia de los avances realizados en la misma. Estos problemas abiertos, que definen futuras líneas de trabajo, son los siguientes:

- La metodología de *benchmarking* de la tecnología de la Web Semántica propuesta en esta tesis únicamente se ha utilizado con tecnología de la Web Semántica. Ya que la metodología es suficientemente general como para considerar cualquier tipo de software, debería ser inmediato utilizarla para hacer *benchmarking* de software de fuera del área de la Web Semántica y analizar si es apropiada para el software en general.
- Otra extensión posible sería analizar si la metodología puede ser también utilizada con procesos o servicios software y no únicamente con productos software. Para ello, nuevas actividades de *benchmarking* deberían realizarse sobre estos tipos de software, analizando los cambios que se requieren en la metodología.

- Con el fin de obtener una metodología general, no se incluyeron técnicas o herramientas concretas que den soporte a las tareas de la metodología. Una forma de completar y mejorar la metodología de *benchmarking* sería identificar las distintas técnicas que se pudieran utilizar en cada una de las tareas del proceso y proporcionar software que de soporte a dichas tareas.
- En las actividades de *benchmarking* descritas en esta tesis, pocas herramientas han participado en comparación con el número de herramientas existentes en la Web Semántica. Un paso futuro sería continuar estas actividades de *benchmarking* cubriendo un mayor número de herramientas.
- En futuras iteraciones de las actividades de *benchmarking*, la evaluación debería actualizarse para ser más exhaustiva, considerando en los conjuntos de pruebas los modelos de conocimiento de las herramientas o de los lenguajes de intercambio al completo, o incluyendo ontologías reales.
- Los experimentos podrían también ser extendidos considerando otros intercambios de ontologías de una herramienta origen a otra destino tales como experimentos de interoperabilidad cíclica (de una herramienta origen a otra destino y a la origen de nuevo) o experimentos de interoperabilidad en cadena (de una herramienta origen a una herramienta intermedia y a una tercera herramienta destino).
- Aunque esta tesis se centra en el problema de la interoperabilidad utilizando un lenguaje de intercambio, otras actividades de *benchmarking* podrían realizarse considerando otras aproximaciones al problema de la interoperabilidad; utilizando otros criterios de evaluación tales como eficiencia, escalabilidad, robustez, etc.; o centrando el *benchmarking* en las necesidades de los usuarios de la tecnología de la Web Semántica.
- Para mejorar la usabilidad de los resultados del *benchmarking*, la mayor mejora sería facilitar el análisis y la explotación de los resultados utilizando una aplicación web pública que realice análisis complejos de dichos resultados. La aplicación IRIBA³⁵ se encuentra actualmente en desarrollo y permite analizar los resultados de la interoperabilidad utilizando RDF(S) en diferentes momentos del tiempo. Con respecto a los resultados de la interoperabilidad utilizando OWL, la herramienta IBSE actualmente genera páginas HTML que resumen dichos resultados. No obstante, sería útil tener una aplicación web que permitiera un análisis dinámico y personalizado de los resultados de la interoperabilidad utilizando OWL.
- También sería útil proporcionar en la herramienta IBSE resultados que son mas fáciles de analizar e incluir visualizaciones específicas para aquellas herramientas cuyos modelos de conocimiento no se corresponden con el lenguaje de intercambio. Para estas herramientas, el análisis de los resultados no es inmediato y a veces información es añadida o eliminada

³⁵<http://knowledgeweb.semanticweb.org/iriba/>

tal y como era deseado por los desarrolladores, pero este funcionamiento correcto es difícil de apreciar con los resultados actuales.

- Otras mejoras técnicas de la herramienta IBSE comprenden cambiar el comparador de ontologías OWL e integrar IBSE con infraestructuras de testeo como JUnit³⁶.

³⁶<http://www.junit.org/>