



UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA



FREE UNIVERSITY OF BOLZANO
FACULTY OF COMPUTER SCIENCE

**EUROPEAN MASTER IN COMPUTATIONAL LOGIC
MASTER THESIS**

Defining a benchmark suite for evaluating the import of OWL Lite ontologies

AUTHOR: Stefano David

TUTORS: Asunción Gómez Pérez (UPM)

Sergio Tessaris (FUB)

CO-TUTOR: Raúl García Castro (UPM)

July 2006

*to Laura and
to my family*

Acknowledgements

This two years Master has been for me a great experience and allowed me to learn a lot from both the courses at the Universities of Bolzano and Madrid and from the working experience at the OEG reserch group in Madrid.

There are several people that helped me in these two years, both when studying for the exams and during the development of this thesis, whose help has been invaluable and allowed me to reach my goal.

I would like to thank my tutor, Prof. **Asunción Gómez Pérez**, director of the OEG group, for her help, support, and suggestions during my working experience.

This thesis has been thoroughly reviewed by my co-tutor, **Raúl García Castro**, whom I am thankful to for the many suggestions received during the development of this work.

I will always be thankful to my fellow **Martin Meraner**, with whom I studied for the exams, for all the help he provided in studying for the courses.

I would also thank all the people at the OEG group, from which I learned a lot, and all the people with whom I spent great time in these two years.

This work has been supported by the IST project Knowledge Web (IST-2004-507482)

Finally, a special thought goes to all my friends who can not partake my joy for completing this study program. I miss you.

Contents

1	Introduction	1
2	State of The Art	3
2.1	Evaluation and benchmarking	4
2.1.1	Benchmark suites	5
2.1.2	Benchmarking methodology	7
2.2	The Semantic Web and its languages	12
2.2.1	The OWL Web Ontology Language	14
2.2.2	Semantic Web tools	18
2.3	Interoperability in the Semantic Web	20
2.3.1	Interoperability approaches in the Semantic Web	20
2.3.2	Evaluating interoperability in the Semantic Web	21
3	Approach	27
3.1	Insufficiencies of the State of the Art	27
3.2	Goals	28
3.3	Scope	29
3.4	General vision for approaching the solution	29
4	Benchmarking OWL interoperability	31
4.1	Plan Phase	31
4.2	Experiment Phase	34
5	The OWL Lite Import Benchmark Suite	37
5.1	Hypothesis	37
5.2	Typology of benchmarks	38
5.3	Benchmarks that depend on the OWL Lite knowledge model.	38
5.3.1	The first approach	38
5.3.2	The second approach	39
5.3.3	The final approach	39
5.4	Benchmarks that depend on the syntax.	48
5.5	Description of the benchmarks	51
5.6	Evaluation criteria	52
5.7	Procedure for executing the benchmark suite	55
5.8	Automation of the execution of the benchmark suite	56

5.9	Towards import benchmark suites for OWL DL and OWL Full	58
6	Conclusions and future work	61
6.1	Lesson learned	61
6.1.1	Benchmarking process	61
6.1.2	Benchmark suite	62
6.2	Future work	63
	Bibliography	65
A	List of benchmarks in the OWL Lite Import Benchmark Suite	71
B	Description of the ontologies in the Description Logics formalism.	83

List of Figures

2.1	The Knowledge Web benchmarking methodology	8
2.2	The Semantic Web layers	13
4.1	The experimentation phases	35
5.1	The first approach to build the OWL Lite Import Benchmark Suite	38
5.2	The second approach to build the OWL Lite Import Benchmark Suite	39
5.3	The OWL DL Import Benchmark Suite	59
A.1	Notation used in the benchmarks of the OWL Lite Import Benchmark Suite	71

List of Tables

2.1	Limited use of OWL vocabulary terms in OWL Lite	16
2.2	Types of tests in the OWL Test Cases.	23
2.3	Summary of the experiments carried out at EON2003.	25
4.1	Ontology development tools capable of importing/exporting OWL	33
5.1	The description of a benchmark of the OWL Lite Import Benchmark Suite.	51
5.2	The different results of a benchmark execution	53
5.3	Fictitious results of executing benchmark ISG03	54
5.4	Restrictions in the use of OWL Lite and OWL DL	60
B.1	Structure of the tables and a sample instantiation	84

1. Introduction

The spreading interest in Semantic Web technologies is leading to the development of an increasing number of tools, each of them providing a different set of functionalities: ontology development tools, ontology repositories, ontology alignment tools, ontology-based annotators, and so on.

As ontologies are widely used in the Semantic Web for representing knowledge, these tools should be able to correctly interchange ontologies between themselves and, therefore, to interoperate. One of the ways of approaching interoperability is to interchange ontologies using an interchange language, which is the one that is considered in this thesis.

Nevertheless, this interchange may not be as straightforward as it seems. Interoperability between tools with different underlying knowledge models using an interchange language requires that these tools are able to translate ontologies from their own knowledge model to the interchange language when they export them, and vice versa when they import them from the interchange language.

In the case of ontology development tools, interoperability is a key issue because these tools use different underlying knowledge models and require frequent ontology interchanges when developing ontologies collaboratively. Hence, assessing the reliability of ontology development tools in the interchange process is a key issue on the way to a better interoperability.

In the Semantic Web area, several benchmarking activities have been organized, focused on different types of Semantic Web technologies. The Knowledge Web Network of Excellence already organized a benchmarking of the interoperability of ontology development tools using RDF(S) as interchange language.

The work performed in this thesis has also been performed within Knowledge Web and continues the benchmarking of the interoperability of ontology development tools. This benchmarking also considers interoperability using an interchange language, but instead of using RDF(S), it uses OWL as interchange language, the language recommended by the World Wide Web Consortium for defining and instantiating ontologies [Smith *et al.*, 2004].

This thesis describes the process followed for the organization of this benchmarking activity and for the definition of the OWL Lite Import Benchmark Suite that is being used in this activity for evaluating the OWL import capabilities of ontology development tools.

This benchmark suite considers the different combinations of ontology components that can be found in ontologies as well as other factors that may influence the interoperability such as the different syntax variants that an OWL serialization in RDF/XML allows.

This thesis is organized as follows.

In Chapter 2 we will introduce the current State of the Art in the domains involved in this thesis, including:

- Evaluation, benchmark suites and benchmarking, and their characteristics, as well as the benchmarking methodology for ontology tools developed in Knowledge Web.
- The Semantic Web, the Web Ontology Language (OWL) considered in the development of this work, and the Semantic Web tools.
- The approaches to evaluate interoperability in the Semantic Web and other similar works, related to the present one.

In Chapter 3 we present the approach we have chosen in the development of this thesis. It contains the existing insufficiencies in the State of the Art, the goals that we want to achieve, and the scope of this work. We will give also give an overview of how we have achieved these goals.

Chapter 4 shows how we adapted the Knowledge Web methodology for benchmarking ontology tools to the development of the OWL interoperability benchmarking.

Chapter 5 is the core part of this thesis. It contains a detailed description of the benchmark suite that we defined for the evaluation of the import of OWL Lite ontologies in ontology development tools.

Finally, Chapter 6 draws the conclusions of our work and discusses some possible evolutions and future developments of this work.

2. State of The Art

This thesis presents a proposal for the evaluation and the improvement of the interoperability of ontology development tools using the Web Ontology Language OWL as interchange language.

The development of this thesis needs a thorough knowledge of the State of the Art in the domains involved. We can recognize three great domains directly covered by this thesis.

Evaluation and benchmarking. Benchmarking is a process that allows not only an evaluation of one or more systems, but also to extract the best practices and to continuously improve these systems.

The Semantic Web, its languages and its tools. The Semantic Web *”provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries”*¹. The idea underlying the Semantic Web is to have data that can be machine-readable and machine-processable, to allow for a better organization of documents and knowledge, a faster search of information, and interoperability among software applications, enterprises, communities of people, industries, and so on.

The Web Ontology Language OWL is the latest step of the W3C for reaching interoperability. One of the purposes of OWL is to provide a standard formalism for representing and interchanging knowledge in the Semantic Web.

Therefore, the Semantic Web needs the development of technologies that can interchange ontologies using these languages.

Interoperability. Interoperability is the ability of two or more systems to interchange information and use this information [IEEE, 1990]. Interoperability is one of the main goals of the Semantic Web: the better two systems can interoperate, the faster information can be processed, interchanged, and (re)used.

This chapter is divided in three sections, according to the three domains identified above. We start with an introduction to the concepts of evaluation and benchmarking and outline the benchmarking methodology that will provide the guidelines for the development of the work presented in this thesis. We will then present the Semantic Web, outline its architecture, describe the OWL language, and define what an ontology development tool is. Finally, we present how interoperability is approached in the Semantic

¹<http://www.w3.org/2001/sw/>

Web and other interoperability evaluations regarding Semantic Web technology that have taken place.

2.1. Evaluation and benchmarking

We give here a brief definition of the terms that will be used throughout the whole thesis. Hence, we define what benchmark, benchmark suite, benchmarking, and software evaluation are. Then, we introduce the characteristics that a benchmark suite should have and finally we describe in detail the benchmarking methodology followed in this thesis.

This section includes content of the Knowledge Web deliverables [Wache *et al.*, 2004] and [García-Castro *et al.*, 2005] that is relevant to our work.

Benchmark. The Standard Glossary of Software Engineering [IEEE, 1991] proposed several definitions of benchmark:

1. *A standard against which measurements or comparisons can be made.*
2. *A procedure, problem, or test that can be used to compare systems or components to each other or to the standards as in 1.*
3. *A recovery file.*

Later, [Sill, 1996] improved the definition at point 2 above, by stating that *a benchmark is a test that measures the performance of a system or subsystem on a well-defined task or set of tasks.*

Benchmark Suite. A benchmark suite is a collection of benchmarks. The properties of benchmarking are discussed in Section 2.1.1

Software evaluation. In [IEEE, 1998], the evaluation of software is defined as *the process of determining the degree to which software possesses a desired combination of quality attributes that fulfill its quality requirements.*

Benchmarking. Among the several definitions available, we have chosen to refer to the following two:

- [Camp, 1989] defined benchmarking as *the search for industry best practices that lead to superior performance.*
- Later, [Spendolini, 1992] refined that definition as *a continuous, systematic process for evaluating the products, services, and work processes of organizations that are recognized as representing best practices for the purpose of organizational improvement.*

According to [Ahmed and Rafiq, 1998], the main characteristics of benchmarking are: a measurement by comparison, a continuous improvement, and a systematic procedure to carry out the benchmarking.

The reason for benchmarking ontology tools instead of just evaluating them is to obtain several benefits from benchmarking that cannot be obtained from tool evaluations. The evaluation of a tool shows us the weaknesses of the tool or its compliance to quality requirements. If several tools are involved in the evaluation, we also obtain a comparative analysis of these tools and recommendations for their users. When benchmarking several tools, besides all the benefits commented, we gain continuous improvement of the tools, recommendations for developers on the practices used when developing these tools and, from these practices, those that can be considered best practices.

There are different types of benchmarking, which are classified in two main ways: a first one according to the type of participants involved and a second one according to the type of the objects analyzed.

The first classification was defined by [Camp, 1989] and foresees four categories of benchmarking:

Internal. This type of benchmarking measures and compares the performances of activities as well as functions and process within one organization.

Competitive. In this benchmarking, products, services and/or business processes are compared with those of a direct competitor.

Functional. This type of benchmarking is similar to the competitive one, but involves more competitors in the same industry.

Generic. It is a benchmarking that aims to a search for the best practices, with no focus on a specific industry.

The second classification was introduced in [Ahmed and Rafiq, 1998] as a complement of the previous one, and later was identified as separate classification in [Lankford, 2000]. It consists of these three types of benchmarking:

Process. This type of benchmarking involves comparisons between discrete work processes and systems.

Performance. It is a benchmarking that involves the comparison and the inspection of performance attributes of products and services.

Strategic. In this benchmarking are involved comparisons of the strategic issues or processes of an organization.

2.1.1. Benchmark suites

Benchmarks are experiments used for measuring the quality attributes of a system. They are usually grouped in benchmark suites and are used in both evaluations and benchmarkings. However, the definition of a **benchmark suite** is not only a group of benchmarks but a collection of several information related to the benchmarks. In particular,

- (a) All the benchmarks that compose the benchmark suite.
- (b) What kind of tools and which of their functionalities can be measured with it.

- (c) The criteria that must be used in the evaluation of the benchmark results.
- (d) A list of the tools and/or the data needed to run the benchmarks.
- (e) The procedure to follow to run the benchmarks.

The main desirable properties of a benchmark suite have been identified by several authors [Bull *et al.*, 1999, Shirazi *et al.*, 1999, Sim *et al.*, 2003, Stefani *et al.*, 2003], and should assist people to develop new benchmarks suites or to assess the quality of existing benchmark suites.

Accessibility. Every people interested in the benchmark suite must have access to it and to every additional component needed for executing it (e.g., documentation, software modules, source code, and so on); the results of the execution of the benchmark suite should also be public, to allow comparisons with other tools.

Affordability. The cost of executing a benchmark suite (in terms of human, hardware and software resources) must be lower than those of having to implement one from scratch and then use it. Also methods for automatizing all or part of the execution of the benchmark suite should be taken into account (e.g., providing components for data collection and analysis).

Simplicity. Every people must be able to understand and interpret the benchmark suite and how it works; also the evaluation of the results and the procedure to use in the experiments should be clear and transparent enough to avoid misinterpretation. Hence, all elements of the suite should have a common structure, use, inputs, and outputs.

Representativity. The benchmark suite must represent the most common actions that a tool usually carries out in its normal use.

Portability. The level of abstraction of the benchmark suite should be high enough so that it can be executed on the largest number possible of environments and systems, thus ensuring the portability of the benchmark suite.

Scalability. The benchmark suite should be parameterized to allow to scale the benchmarks with varying input rates. It should also scale to work with tools or techniques at different levels of maturity. It should be applicable both to research prototypes and commercial products.

Robustness. The benchmark suite must consider unpredictable behaviours of the environment, and should not be sensitive to factors not relevant to the study. When running the same benchmark suite several times on a given system under the same conditions, the results obtained should not change.

Consensus. The benchmark suite must be developed by experts that provide their knowledge about the domain, and are able to identify the key problems. It should also be assessed and agreed on by the whole community.

2.1.2. Benchmarking methodology

A benchmarking methodology provides a set of guidelines that shall be followed in benchmarking activities. The methodology that we describe in this section has been developed from methodologies from different areas such as business community benchmarking, software experimentation, and software measurement and it is being used within the IST-2004-507482 Network of Excellence Knowledge Web² for various benchmarking initiatives.

The work presented in this thesis is developed within Knowledge Web and the benchmarking methodology used in this work, which is described in the remainder of this Section, is the one developed in Knowledge Web [García-Castro *et al.*, 2005]. The instantiation of this general methodology to our own case is described in Chapter 4.

This benchmarking methodology includes the main phases that must be carried out when benchmarking ontology tools, with a definition of the tasks to perform in each of them, considering its inputs, outputs, and actors. Therefore, the methodology can be used in a twofold manner: to assist in carrying out benchmarking activities or to know, at a certain point in time, which is the actual progress of a benchmarking activity.

This methodology considers that the benchmarking activity is focused on software products, in this specific case, ontology development tools. Benchmarking can also deal with methods or processes, but we have limited the scope of the methodology for a better understanding.

Although this methodology has been developed for benchmarking activities in Knowledge Web to cover all the benchmarking activities developed and supported in this Network of Excellence, it is intended to be as general as possible, in order to allow other people and institutions not involved in the Network of Excellence (for example the industry or other academical projects) to take advantages from it and to use it.

The actors of the benchmarking process

Before presenting the different tasks that must be carried out in the benchmarking process, we identify the actors that take part in it:

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 activity in the organization and in obtaining the management approval to perform it. After receiving the approval for starting the benchmarking process, he will coordinate all the remaining tasks and he will also take part to the experiments and to their analysis.

Organization management. The management of the organization plays a key role in the benchmarking process, as it must approve the benchmarking activity and the changes

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

in the tool and in the organization itself that might result from the benchmarking process. It must also assign resources for the benchmarking to take place and integrate the benchmarking planning into the whole organization planning.

Benchmarking team. The benchmarking team is composed by the members of the organization that, after the organization management approval, will be responsible for performing most of the remaining benchmarking process. Its members are usually selected among the developers of the tool, managers of the organization, ontology engineers, end users, and by the benchmark initiator.

Benchmarking partners. The benchmarking partners are the organizations that take part in the benchmarking activity. They play an active role in the whole benchmarking process and can suggest, criticize, pose questions, express particular needs: in practice, they can help in the improvement of the benchmarking process. Moreover, all the partners must agree on the steps to carry out during the benchmarking.

Tool developers. The developers of the tools are the ones that will implement the necessary changes in the tool, considered for the benchmarking, in order to improve it, following the results of the benchmarking activity and the recommendations emerged from them. Some of them may also be part of the benchmarking team and, in this case, care must be taken to minimize bias.

The benchmarking process

The benchmarking methodology for ontology tools proposed in Knowledge Web, is a continuous process that is composed of a benchmarking iteration that should be repeated indefinitely. Figure 2.1 presents the three phases that compose the methodology with the tasks to be performed in each phase. This methodology is described in detail below.

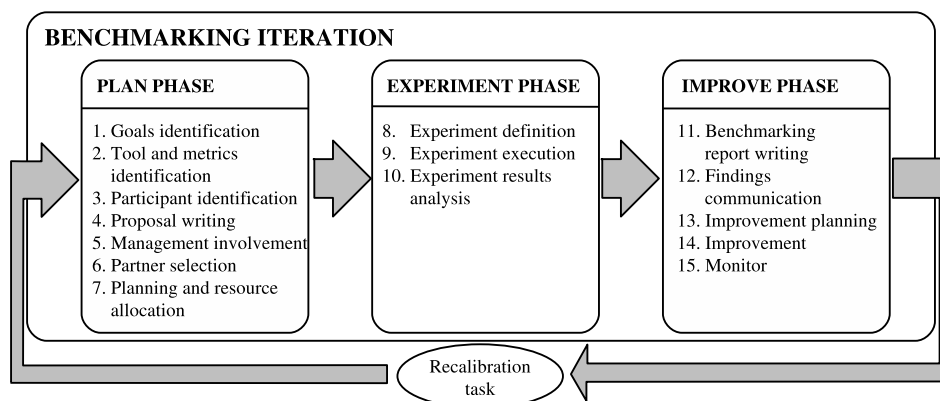


Figure 2.1.: The Knowledge Web benchmarking methodology

Plan Phase

The Plan phase is composed of the sequence of tasks that must be performed to prepare the benchmarking proposal, to find other organizations that participate in the benchmarking activity, and to plan and organize the benchmarking. The tasks to be performed in this phase are:

Goals identification. The benchmarking process starts in an organization in which one of its members, who becomes the benchmarking initiator, is aware of the need for benchmarking. This need for benchmarking varies from one organization to another and is highly related to the desired goals of the benchmarking process. The main goals to achieve [Sole and Bist, 2005, Wiremann, 2003] by starting the benchmarking process are:

- To assess the performance and improve the quality of the products and the processes of the organization.
- To compare the products and processes of the organization with those of the best organizations and to reduce or close the gap with the competitors.
- To understand in details the practices that lead to the creation of better products and processes and establish standards set by the industry or to create them after the analysis of the best organization.
- To increase the satisfaction of the customers for the products of the organization.

The benchmarking initiator has to identify also the benefits brought to the organization by carrying out the benchmarking process and has to estimate the costs of the whole benchmarking process.

Tools and metrics identification. The benchmarking initiator analyzes all the tools developed in his organization and identifies their functionalities and weaknesses that shall be improved, producing a document that summarizes this findings. Moreover, he also selects which tools, and which of their functionalities, should take part in the benchmarking and define the criteria to evaluate the results of the experiments.

Participant identification. The benchmarking initiator contacts the members of his organization, which are involved with the selected tool(s) and selects among them the benchmarking team, that will perform most of the remaining task in this and in the other phases. The members of the benchmarking team can be developers, managers, users or customers (i.e., they can also be external to the organization) and should have experience with the tool and its relevant functionalities. The benchmarking initiator can be part of this team.

Proposal writing. The benchmarking team collects all information produced in the previous tasks and produces a document with the benchmarking proposal, which will be used as a reference in the rest of the benchmarking process. As this document is intended for heterogeneous readers (developers, managers, management and so on), it should be readable and understandable by all of them.

Management involvement. The benchmarking initiator presents the document with the proposal, produced in the previous task, to the management of its organization, in order to receive support for the development and continuation of the process.

Partner selection. The benchmarking team should look for other tools that might take part to the benchmarking and collect information about them and the organizations that develop them. It should also define criteria to identify other tools suitable for the benchmarking, which are:

- Importance of the tool in the community or in the industry.
- Use of the latest available technologies.
- Wide use of the tool by the community.
- Public availability (e.g., as freeware or under a GPL-like³ license).

The organizations that develop the identified tools, should be contacted to check their interest in participating to the benchmarking. All the organizations that accept to participate become benchmarking partners and shall set up a benchmarking team and present a benchmarking proposal to their management for approval. If this proposal is modified, the changes are proposed to other partners and the original proposal must be modified as well. At the end all partners have to agree on the benchmarking proposal and use it as reference in the remaining tasks.

Planning and resource allocation. The managers and the benchmarking teams of each involved organization define a plan for the remaining tasks of the process, taking into account all efforts needed during the whole process: time, human resources, financial, and so on.

Experiment Phase

The Experiment phase is composed of the sequence of tasks related with the experimentation on the different tools that are considered in the benchmarking. The tasks to be performed in this phase are:

³<http://www.gnu.org/licenses/licenses.html#GPL>

Experiment definition. The benchmarking teams define a set of experiments that will be carried out on all the tools. These experiment must be designed to evaluate the functionalities of the tools, as reported in the benchmarking proposal, and they must evaluate in an objective and reliable way the tools. Moreover, also a planning and a procedure for executing the experiments should be provided.

Experiment execution. Each benchmarking team carries out the experiments on its tool. All the data obtained will be collected and written in a common format to ease the analysis of the results.

Experiment results analysis. The results of each tool and the practices that lead to them are analyzed and compared; the benchmarking teams should also identify the best practices, if there are any. The benchmarking teams will write a technical report containing all the findings obtained in the experiment phase.

Improve Phase

The Improve 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. The tasks to be performed in this phase are:

Benchmarking report writing. The benchmarking teams write a report targeted to all the people involved in the benchmarking process (developers, managers, users, and so on) but also to external people and organizations. This report should clearly describe the benchmarking process followed, the results of the experiments, the best practices found, and provide recommendations from the benchmarking teams to improve the tools.

Findings communication. Each benchmarking team organizes one or more meetings that possibly involve more organizations, in order to communicate the results of the benchmarking. The purposes of these meetings are:

- To obtain feedback and improvement recommendations on the benchmarking process and the results.
- To obtain support and commitment from the organizations for implementation in the tools of the improvement recommendations.

Improvement planning. This task, together with the next two (i.e., *Improvement* and *Monitoring*), forms a cycle that must be performed separately in each organization. It is in these tasks where each organization benefits from the results obtained in the benchmarking.

The management and the benchmarking team of each organization identify, from the benchmarking report and from the monitoring report (see the *Monitor* task below), the

changes needed for improving the tools, a mechanism to ensure the accomplishment of the improvements, and define a plan for the improvement, which takes into account all possible factors that may influence it: time, human resources, financial, and so on.

Improvement. Before the developers of the tools implement the changes defined in the previous task, they must assess the actual state of the tool by means of the mechanism identified in the previous task; after the implementation, they have to assess again the state of the tool, and compare the new state with the one the tool had before the improvements.

Monitor. Each benchmarking team provides means for monitoring its tool (monitoring tools, an updated version of the benchmark suites and so on); the developers monitor regularly their tool and write a monitoring report containing the results. If the monitoring results in a need for new improvement, a new cycle of *Improvement planning*, *Improvement* and *Monitoring* can be started, using the monitoring report as reference.

Recalibration Task

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 it. This way, the organization gets improvement not just in the tools, but also in the benchmarking process. This recalibration is needed because both the tools and the organizations evolve over time.

2.2. The Semantic Web and its languages

In the last years there has been an impressive increase of the World Wide Web technologies and of their use. The Web is today the largest collection of data ever existed, but its organization is rather inefficient. The idea underlying the Semantic Web is to provide a framework that allows data to be shared and reused by different applications, organizations, and people⁴. It is meant to have machine-readable data, i.e., to allow computers to manage data in a more efficient way, providing a faster search of information, and a more organized knowledge and knowledge management.

Figure 2.2 shows the Semantic Web layer, i.e., the foreseen architecture of the Semantic Web, as imagined by its author, Tim Berners Lee. We outline here briefly the building blocks of the Semantic Web. A deeper discussion on the Semantic Web layers can be found online⁵.

Unicode character. Computers only deal with numbers, hence a string of characters has to be encoded in numbers before a computer can process it. Unicode⁶ provides a

⁴<http://www.w3.org/2001/sw/>

⁵<http://www.w3.org/DesignIssues/Semantic.html>

⁶<http://www.unicode.org>

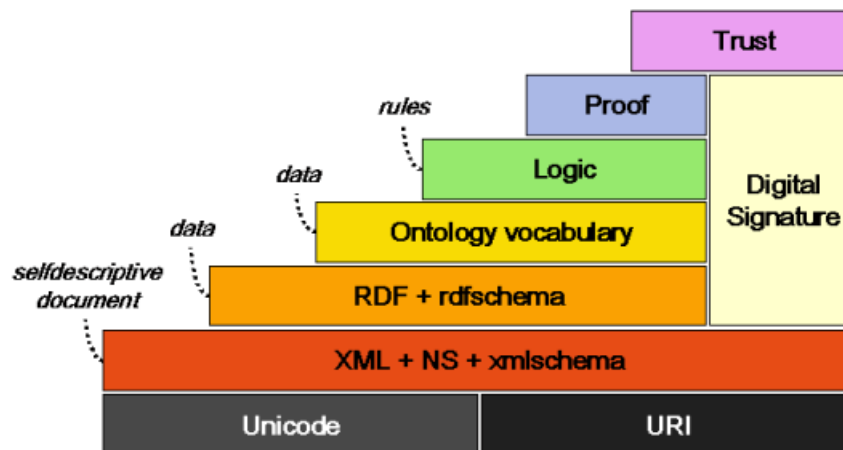


Figure 2.2.: The Semantic Web layers

standard character encoding for almost all languages of the world, including punctuation symbols.

URI references. Each resource on the web can be identified by a unique string of characters. This string is called *Uniform Resource Identifier (URI)*, and is defined in [T.Berners-Lee and M as a compact string of characters for identifying an abstract or physical resource.

XML and XML Schema. The eXtensible Markup Language (XML⁷) is a standard data format to exchange data on the web, derived from SGML (Standard Generalized Markup Language). XML Schema⁸ gives the opportunity to define the content, the structure, and the semantics of an XML document.

RDF and RDF Schema. The RDF language allows to represent information about resources (i.e., metadata) in the World Wide Web, while RDF Schema provides a formal semantics to relate resources and metadata, and inference rules for allowing reasoning over them.

Ontology Vocabulary. Languages like OIL⁹, DAML+OIL¹⁰, and OWL¹¹ allow the use of ontologies in the World Wide Web, providing a large vocabulary to define complex properties (e.g., transitive or functional properties, cardinality and value restrictions on properties) and complex classes (e.g., classes defined by union or intersection of other classes).

⁷<http://www.w3.org/XML/>

⁸<http://www.w3.org/XML/Schema>

⁹<http://www.ontoknowledge.org/oil/>

¹⁰<http://www.daml.org/>

¹¹<http://www.w3.org/2004/OWL/>

Logic, Proof and Trust. These layers and the following are not yet well defined, and are subject of investigation. The logic layer allows the definition of rules on top of ontology, for adding additional expressiveness to the ontology layer.

The Proof layer will allow the execution of the rules defined in the Logic layer. The results of the execution of rules are evaluated together with the Trust layer, to check whether the given proof can be trusted or not.

Digital Signature. The conjunction of the layers from the RDF layer to the Proof layer will allow to digitally *sign* a document. In other words, it will be possible for the author of a document to be recognized as such and to check for changes made to that document by unauthorized people.

2.2.1. The OWL Web Ontology Language

In this Section we present the OWL Web Ontology Language, which is a Technical Recommendation of the W3C Consortium¹² since 2004. We outline the differences among its three sublanguages, then we present the vocabulary of OWL and the limitations that OWL Lite has in using this vocabulary, the different syntaxes of OWL, and finally some OWL considerations related to this work.

The OWL vocabulary

The vocabulary of OWL is composed by several vocabulary terms, whose purpose is to increase the expressiveness of the RDF(S) vocabulary and to extend the RDF(S) capability to describe classes and properties. The most relevant improvements introduced by the OWL vocabulary are the possibility to express disjointness or equality among different classes or among different individuals, to express cardinality constraints on properties, and to further specify logical characteristics of properties (e.g., transitivity or symmetry). We do not report the whole OWL vocabulary here, as it can be found online¹³, along with examples and illustration of its use.

Moreover, the use of some vocabulary terms of the RDF(S) vocabulary is forbidden when building OWL Lite or OWL DL ontologies, as their semantics conflict with the correspondent OWL vocabulary terms.

The sublanguages of OWL and their limitations

OWL is composed by three layered sublanguages, which are: OWL Lite, OWL DL, and OWL Full, in order of increasing expressiveness. We describe here the most relevant

¹²<http://www.w3.org/>

¹³<http://www.w3.org/TR/owl-features/>

differences between the semantics and the use of vocabulary terms in the three species of OWL. An exhaustive description of these differences between them can be found online¹⁴.

OWL Full can use all the OWL and RDF(S) vocabulary terms with no restrictions; as it is a proper superset of RDF(S), it gives the user the freedom to extend the RDF(S) vocabulary with the OWL vocabulary terms and also to augment the meaning of both vocabularies. In particular:

- The vocabulary terms `rdfs:Class` and `owl:Class`, `rdfs:Resource` and `owl:Thing`, and `rdf:Property` and `owl:ObjectProperty` are equivalent.
- The built-in `owl:DatatypeProperty` class is a subclass of the built-in `owl:ObjectProperty` class.
- There is no disjunction among classes, datatypes, datatype properties, object properties, annotation properties, ontology properties (i.e., import statements and versioning properties), individuals, data values, and the built-in vocabulary

OWL DL can use all the OWL vocabulary terms, but sets a number of limitations on their use: OWL DL cannot modify its vocabulary and forbids the use of some RDF(S) constructors. In particular:

- Classes, datatypes, datatype properties, object properties, annotation properties, ontology properties, individuals, data values, and the built-in vocabulary are pairwise disjoint. This implies, for example, that a class can not be treated as an individual and that OWL DL can not modify its vocabulary, as OWL Full can.
- A datatype property can have as object only a literal value (i.e., an instance of `rdfs:Literal`) and a literal can never be subject of a property, hence datatype properties can neither be specified as symmetric, as transitive, as inverse of another datatype property, nor as inverse functional.
- The use of a large part of the RDF(S) vocabulary is forbidden in OWL DL, for example `rdfs:Class` or `rdf:Property`.
- Transitive properties can not have cardinality constraints, neither local (i.e., using one of the terms `owl:cardinality`, `owl:minCardinality`, or `owl:maxCardinality`), nor global (i.e., using a functional or an inverse functional property).
- All axioms must be well formed and form a tree-like structure.

OWL DL has been named this way because its semantics resembles the one of the *SHOIN(D)* Description Logic [Horrocks and Patel-Schneider, 2004].

¹⁴<http://www.w3.org/TR/owl-ref/#Sublanguage-def>

OWL Lite cannot use some of the OWL vocabulary terms and has limitations in the use of others. Besides the limitations of OWL DL, the OWL vocabulary terms that can not be used in OWL Lite are:

- `owl:hasValue`, used to specify the filler of a property (i.e., the value that a property can have in its range).
- `owl:unionOf`, used to define classes as union of other classes.
- `owl:disjointWith`, used to define that two or more classes have no instances in common. Used in conjunction with `owl:unionOf`, allows to create a partition of a class.
- `owl:complementOf`, used to define classes as complement of other classes.
- `owl:DataRange`, used to define enumerated datatypes.
- `owl:oneOf`, used to describe classes explicitly as collections of individuals. In conjunction with `owl:DataRange` allows to define collections of datatypes.

The vocabulary terms of OWL that have a limited use in OWL Lite are shown in table 2.1.

Vocabulary Term	Use in OWL Lite
<code>owl:cardinality</code>	limited to a value of 0 or 1
<code>owl:minCardinality</code>	limited to a value of 0 or 1
<code>owl:MaxCardinality</code>	limited to a value of 0 or 1
<code>owl:intersectionOf</code>	only with lists of class names or restrictions
<code>owl:equivalentClass</code>	subject must be class names and object must be class names or restriction
<code>owl:subClassOf</code>	subject must be class names and object must be class names or restriction
<code>owl:allValuesFrom</code> <code>owl:someValuesFrom</code>	object must be class names or datatype names
<code>rdf:type</code>	object must be class names or restrictions
<code>rdfs:domain</code>	object must be class names
<code>rdfs:range</code>	object must be class names or datatype names

Table 2.1.: Limited use of OWL vocabulary terms in OWL Lite

These limitations result in a fewer expressiveness and complexity of OWL Lite, than those provided by OWL DL. OWL Lite is equivalent to the *SHIF(D)* Description Logic [Horrocks and Patel-Schneider, 2004] and is intended for an easy implementation and as a bridge for migrations from existing RDF(S)-based technologies (e.g., ontologies, software applications and so on) to OWL.

The different OWL syntaxes

We can recognize two types of OWL syntaxes. On the one hand the OWL abstract syntax, which consists of a set of productions that define the grammar of the OWL language. These productions are divided in *Ontologies* and *Facts*, which are common to OWL Lite and OWL DL, and by different *Axioms* for OWL Lite and OWL DL. The *Axioms* are further divided into *class*, *restriction* and *property* axioms, common to OWL Lite and OWL DL, and by *description* axioms for OWL DL.

We do not report all the productions here, as they can be found online¹⁵, they will be presented in Section 5.3 when we will describe how we used them in the definition of the benchmarks of the OWL Lite Import Benchmark Suite.

On the other hand, there are three syntaxes commonly used to serialize OWL ontologies: Notation3¹⁶, Turtle¹⁷, and RDF/XML¹⁸. Notation3 and Turtle are very similar and were designed to write ontologies in human readable form, whereas RDF/XML is intended to write ontologies in machine readable form, for a faster machine processability.

Other considerations to take into account

Since the work presented in this thesis is likely to be extended at some point in the future, and since the semantics of OWL is provided by the Description Logics formalism, it is worth presenting some aspects of this formalism that might confuse or lead to errors.

The three important aspects to point out are: the differences between the Open and Closed World Reasoning, the difference between primitive and defined classes, and the range and domain constraints.

Open World Reasoning

The Closed World Reasoning and the Open World Reasoning (or Open World semantics, see [Baader *et al.*, 2003]) represent two different approaches in how to evaluate *implicit* knowledge in a knowledge base. Implicit knowledge are information contained in the knowledge base, but not explicitly stated.

The former approach is the one used in database, whereas the latter is the one used in Description Logics and therefore also in OWL.

The difference in the behaviour is usually clarified by a comparison between the structure of a knowledge base $\Sigma=(\mathcal{T}, \mathcal{A})$, where the TBox \mathcal{T} represents the terminology of the ontology (i.e., the structure of the classes and the properties) and the ABox \mathcal{A} represents the assertions in the ontology (i.e., instances of classes and relations among them), and a relational database where its schema represents \mathcal{T} (i.e., its tables and structure) and its tuples represent \mathcal{A} . The difference between them is given in terms of interpretations.

On the one side, in the Description Logics formalism, we can have different interpretations of Σ , on the other one, in databases, we have the only one interpretation that a

¹⁵<http://www.w3.org/TR/owl-semantics/syntax.html>

¹⁶<http://www.w3.org/DesignIssues/Notation3.html>

¹⁷<http://www.dajobe.org/2004/01/turtle/>

¹⁸<http://www.w3.org/TR/rdf-syntax-grammar/>

database can have. Hence, while in a database what is not explicitly stated in one tuple is interpreted as “*negative*” or false knowledge, in knowledge bases it is interpreted only as absence or incompleteness of knowledge.

The most known and subtle example of difference between the Open and Closed World Reasoning was originally created and presented in [Donini *et al.*, 1996] and lately named the *Little house problem* in [Damiani *et al.*, 2005].

Primitive and Defined classes

There are two possibilities to describe a class in OWL: either as a *primitive* class or as a *defined* class.

- A **primitive** class is described by a set of *necessary* conditions that a resource must satisfy. This is a partial class in the OWL Abstract Syntax, expressed in OWL by defining it as subclass of another class, of combinations of classes, or of combinations of classes and restrictions, using the `rdfs:subClassOf` vocabulary term.
- A **defined** class is described by a set of *necessary and sufficient* conditions that a resource must satisfy. This is a complete class in the OWL Abstract Syntax, expressed in OWL by defining it as `owl:equivalentClass` of another class, of combinations of classes, or of combinations of classes and restrictions, using the `owl:equivalentClass` vocabulary term.

As stressed in different papers (for example in [Rector *et al.*, 2004]), this difference is essential in the inference process. The key point is to understand that a process of inference will not be able to classify an individual as an instance of a primitive Class.

Range and domain constraints

These constraints do not indicate, in OWL, the type of an individual in the range or in the domain of a property, but they are used in reasoning processes to determine the type of instances related by a property, i.e., given classes as domain or range of a property, then individuals related by means of this property are inferred to be instances of those classes.

If there are two or more classes in the domain (or in the range) constraints, than the instance is classified as instance of the intersection of all those classes.

2.2.2. Semantic Web tools

We outline here the main characteristics of Semantic Web tools. We have divided them in 4 categories: knowledge model, knowledge representation languages, connection with reasoners, and storage of ontology.

Knowledge model. The internal knowledge model is the core of a Semantic Web tool: it contains the components that can be modelled to create a knowledge base (i.e., an ontology). Some Semantic Web tools may support more than one knowledge model. The most used knowledge models used today for ontology development tools and relevant for

the Semantic Web are frame-based or Description Logics-based, depending on the type of logic implemented.

Frame-based representation [Fikes and Kehler, 1985] allows the description of concepts as *frames*, which can also have additional information attached to them. Without going into much details, frames can be either *Classes* or *Slots*: a class represents a set of individuals in the domain and a slot represents a property of a class. Classes are organized in (hierarchical) taxonomies and may have instances, i.e., single individuals that satisfy the property of a class. Classes, slots and individuals can be used to set up a knowledge base, on which there is also the possibility of reasoning by means of rules. Earlier frame-based systems did not have a clear underlying semantics, hence many efforts have been devoted to fill this hole. The results of these efforts is the *F-Logic* (Frame Logic) formalism [Kifer *et al.*, 1990].

Description Logics (DLs) [Baader *et al.*, 2003] is a formalism that represents an evolution of the frame-based logic representation. The two main components of a DLs knowledge base are the TBox and the ABox: the former contains the vocabulary (i.e., the *terminology*) of the knowledge base, while the latter contains *assertions* about individuals using the vocabulary defined in the TBox.

The TBox contains (*atomic*) *concepts*, that are collections of individuals, and (*atomic*) *roles*, that denote binary relationships between individuals. Atomic concepts and roles can be composed to define complex classes and Roles. The inference processes and the reasoning support for DLs has been thoroughly investigated in the last few years, where algorithms has been discovered even for very expressive DLs.

Knowledge Representation languages. There are several languages in which an ontology can be represented, the most popular today in the field of Semantic Web, are RDF(S)¹⁹, OWL²⁰ and their ancestors, OIL²¹ and DAML+OIL²². However, outside the Semantic Web, other language are used for knowledge representation, for example KIF²³ (Knowledge Interchange Format), a computer-oriented language whose purpose is to facilitate sharing and interchanging of knowledge among different systems.

Moreover, many Semantic Web tools use an own language to store ontologies and meta-data about them.

Connection with reasoners. A reasoner is a tool that takes a knowledge base as input, computes some inference process and gives an output depending on the inference process. An ontology development tool usually has the facility to connect with an external reasoner, for helping developers in building ontologies and also for computing inference processes on them. The output of a reasoner can be either:

- The original knowledge base augmented with implicit knowledge, i.e., knowledge contained in the knowledge base but not explicitly stated.

¹⁹<http://www.w3.org/RDF/>

²⁰<http://www.w3.org/2004/OWL>

²¹<http://www.ontoknowledge.org/oil>

²²<http://www.daml.org/>

²³<http://www.ksl.stanford.edu/knowledge-sharing/kif/>

- The answer to a query explicitly asked by the user.
- A consistent ontology or the notification of the inconsistency of the ontology, i.e., that in the ontology there are contradictory or unsatisfiable concepts.

Storage of the ontology. All the tools can store ontologies in different ways: in some relational database, in files on the hard disk, in the World Wide Web, or in ontology repositories. Every tool can also serialize ontologies using one or more of the languages cited above or some own format.

These characteristics are common to all types of Semantic Web tools and therefore also for ontology development tools, on which the work presented in this thesis is focused.

An ontology development tool is defined in [Gómez-Pérez *et al.*, 2003] as *a tool that can be used to build a new ontology from scratch. In addition to the common edition and browsing functions, these tools usually give support to ontology documentation, ontology export and import to/from different formats and ontology languages, ontology graphical edition, ontology library management, etc.*

2.3. Interoperability in the Semantic Web

Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE, 1990].

In this section we first provide a description of the approaches that can be adopted to achieve interoperability in the Semantic Web and then presents three initiatives similar to the one presented in this thesis.

2.3.1. Interoperability approaches in the Semantic Web

The tradeoff between expressiveness and reasoning in a formalism (i.e., the more expressive is a language, the higher is the complexity of reasoning on it), identified in [Brachmann and Levesque, 1985], leads to different approaches in the interoperability, depending on the expressiveness required.

There are different ways to achieve interoperability between Semantic Web technology using an interchange language, which are formally defined in terms of the languages used to represent knowledge in [Euzenat and Stuckenschmidt, 2003].

In this section we will use the following conventions. L, P are languages, defined as a set of expressions that can be generated from a set of atomic terms and a set of constructors, r is a term or a set of terms in the language, $\tau, (\tau : 2^L \rightarrow 2^{L'})$ is a transformation from a language L to a language L' And $L \preceq_p L'$ denotes the existence of a transformation τ from L to L' that satisfies a property p .

1. The Mapping Approach. This approach requires to relate expressions in one source language to a target language. Formally, this approach is defined by:

$$\exists \tau : (\forall r \subseteq L, \tau(r) \subseteq L') \quad (2.1)$$

In other words, there is a function τ that maps an expression r in the source language L to an expression $\tau(r)$ in the target language L' . This is easy to achieve, and provides a direct interface to interchange knowledge represented in two different languages, but has the drawback that requires a mapping between each pair of languages, if more languages are used, and is therefore not reusable.

- 2. The Pivot Approach.** This approach requires that an expression in the source language is expressed into a common, intermediate language (the pivot language) and transformed from this into the target language. Formally:

$$\exists! P : (\forall L, L \preceq_p P) \quad (2.2)$$

Informally, there is a unique pivot language P that includes (i.e., is more expressive than) all other languages L . In this approach, the choice of the pivot language P is substantial: it has to be expressive enough for containing all other languages, in order to preserve semantics: i.e., P must have the ability to encode all other languages: if P can not express a term r in one language L , the consequence is a loss of knowledge when trying to interchange knowledge from a language L to a language L' . This approach solves the complexity issue of the mapping approach, as it needs a considerable lower number of transformation compared to the mapping approach. However, there might be some loss of knowledge when translating a set of terms r from the pivot language P to a less expressive language L .

- 3. The Layered Approach.** The necessity of a very expressive language on the pivot approach is overcome here using a layered architecture of languages with increasing expressiveness, included one in the other. In this approach, written formally in 2.3, there is no unique pivot language, and every language L_i can be translated in a language L_j higher in the hierarchy, without loss of knowledge.

$$\forall i, j : (i \leq j \Leftrightarrow L_i \preceq_p L_j) \quad (2.3)$$

- 4. The 'Family of Languages' Approach.** It is a generalization of the pivot and layered approaches and requires neither a fixed pivot language nor a fixed layering of languages. Formula 2.4 shows that this approach requires a partial ordering (instead of a total ordering) of the layered languages and, for every pair of languages L, L' , a third language L'' that contains both. For this reason, and since the third language L'' is difficult to know and to define a priori, this approach is difficult to manage.

$$\forall L, L' \exists L'' : (L \preceq_p L'' \wedge L' \preceq_p L'') \quad (2.4)$$

2.3.2. Evaluating interoperability in the Semantic Web

To the best of the authors' knowledge, in the recent past, three other initiatives similar to the one presented in this thesis have been taken place. Two of them were focused on

the evaluation of the interoperability of Semantic Web technology (the EON 2003 Workshop and the RDF(S) Interoperability Benchmarking), while the third one (the OWL Test Cases) was focused on the import of OWL ontologies into Semantic Web tools. We introduce them here in chronological order from the first describing their main characteristics.

The OWL Test Cases

The OWL Test Cases [Carroll and Roo, 2004] were developed by the W3C Web Ontology Working Group. Their first official version dates back in October 2002²⁴, as a W3C Working Draft. They are a collection of simple ontologies that show examples of the correct use of the OWL language and of its vocabulary terms²⁵ and also show how the Web Ontology Working Group solved some issues raised during the development of OWL.

Table 2.2 summarizes the types of tests defined in the OWL Test Cases and describes them briefly. In that table, a document is an ontology saved in a RDF/XML file.

Although the OWL Test Cases do not represent a benchmark suite for assessing interoperability, they might be used for the evaluation of the OWL importers of the ontology development tools, as they require to import one document in the tool considered.

The W3C Consortium maintains a web page²⁶ that lists the results obtained by several applications when tested against the OWL Test Cases.

The characteristics of the OWL Test Cases are:

Target. The OWL Test Cases were defined for any tool that implements OWL ontologies.

Purpose. The purposes of the OWL Test Cases are to check whether a tool (not necessarily an ontology development tool but every tool that can manage OWL ontologies) correctly deals with the OWL language, to clarify the formal meaning of the OWL constructors, and to show examples of their use. They are also intended as a guideline for the developers of new tools, that want to implement the OWL language.

Language. There are test cases defined for all of the three sublanguages of OWL (Lite, DL and Full).

Syntax. The OWL Test Cases assume that the ontology is written in correct RDF/XML and do not have tests for checking whether the tools can deal with the variants allowed by the RDF/XML syntax.

Valid ontologies. The OWL Test Cases contain sets of invalid ontologies, which use invalid namespaces or incorrect OWL vocabulary terms. These tests are intended for helping in the migration from DAML+OIL ontologies to OWL ontologies.

²⁴<http://www.w3.org/TR/2002/WD-owl-test-20021024/>

²⁵The OWL vocabulary terms are also called *constructors*.

²⁶<http://www.w3.org/2003/08/owl-systems/test-results-out>

Type of Test	Description
Incorrect Use of the OWL Vocabulary	The tool must import a document that includes a vocabulary term that does not belong to OWL.
Entailment	The tool must import a document that contains an ontology from which it is possible to entail the conclusions contained in a second document.
Non-Entailment	The tool must import a document that contains an ontology from which it is not possible to entail the conclusions contained in a second document.
True Tests	These are a special case of entailment tests, in which there is no ontology to import, but the conclusions contained in a document can still be entailed.
OWL for OWL	These are a special case of true test, in which the conclusions contained in a document follow directly from the OWL Full semantics.
Consistency	The tool must import a document that contains a consistent (w.r.t. the OWL semantics) ontology.
Inconsistency	The tool must import a document that contains an inconsistent (w.r.t. the OWL semantics) ontology.
Import Entailment	The tool must import a document that imports one or more documents. All together they entail the conclusions contained in another document. The imported documents are contained in the imports closure of the document that imports them, i.e., they form subparts of the premises.
Import Level	The tool must import a document that imports another document, which is written in a different OWL sub-language.

Table 2.2.: Types of tests in the OWL Test Cases.

Semantics. As the OWL Test Cases are targeted to any kind of technology that can deal with OWL, a number of tests involving semantics are included. There are entailment and non-entailment tests, consistency and inconsistency tests, and import test. Furthermore, this kind of tests involve the simultaneous import of more than one RDF/XML document.

Execution of the experiments. Each type of test shown in Table 2.2 is composed by one or more documents. One document must be imported into the tool considered and then, depending on the type of test, one of the following situations happen.

- The tool in which the document is imported should notify an error (i.e., in tests for incorrect use of the OWL vocabulary and for inconsistency).
- The tool shows a valid ontology (i.e., in true tests, OWL for OWL, consistency, and

import level tests).

- The knowledge shown by the tool after the import process must be equivalent to that contained in another document that is part of the test (in all the remaining types of tests).

EON 2003 Workshop

The central topic of the Second International Workshop on Evaluation of Ontology-based Tools (EON 2003) was the evaluation of the interoperability of ontology development tools.

In the experiments organized in this workshop, the participants were asked to model an ontology in their ontology development tools and to perform different tests for evaluating the import and export capability of the tool and its interoperability capabilities with other tools using an interchange language [Corcho and Sure, 2003].

The characteristics of this experimentation were the following:

Methodology. The activity had no underlying methodology, hence it was not really a benchmarking activity but only an evaluation of the interoperability of ontology development tools.

Language. There was no constraint in the choice of the interchange language. The participants could choose the language they preferred (e.g., RDF(S), DAML+OIL, OIL, and so on) and OWL was used as interchange language only in few experiments. Moreover, OWL had not yet reached the status of *Technical Recommendation* of the W3C Consortium, hence these experiments are outdated and may not reflect the actual specification of OWL.

Procedure. There was no fixed procedure that all the participants should follow to perform the experiments. Hence, the interpretation of the results of the interoperability experiments was localized to the single tools. As a consequence, neither general recommendations could be provided (e.g., for the improvement of the ontology development tools), nor best practices could be extracted from the experiments.

The experiments. During the experiments it was suggested to interchange the traveling domain ontology developed in the previous EON workshop²⁷ (EON2002). Different types of experiments were performed in various tools by the participants: a summary of them is presented in Table 2.3 and below:

1. Interoperability of Protégé 2.0 beta and OilEd [Calvo and Gennari, 2003]. In this experiment, an ontology (in the osteoporosis domain) was developed in both tools, then exported to RDF(S) and imported in the other tool.

²⁷<http://km.aifb.uni-karlsruhe.de/ws/eon2002/>

Experiment	From tool	To tool	Interchange Language
1.	Protégé 2.0 Beta	OilEd	RDF(S)
	OilEd	Protégé 2.0 Beta	RDF(S)
2.	Protégé-2000 1.8	WebODE 2.0	RDF(S)
	WebODE 2.0	Protégé-2000 1.8	RDF(S)
3.	DOE	OilEd	RDF(S)
	DOE	OntoEdit	RDF(S)
	DOE	Protégé-2000	RDF(S), OWL
	DOE	WebODE	RDF(S), OWL
	OilEd	DOE	RDF(S) OWL
	OntoEdit	DOE	RDF(S)
	Protégé-2000	DOE	RDF(S), OWL
	WebODE	DOE	RDF(S), OWL
4.	KAON	SemTalk	DAML
	OilEd	SemTalk	DAML
	OntoEdit	SemTalk	DAML
	Protégé	SemTalk	RDF(S)
	WebODE	SemTalk	RDF(S)

Table 2.3.: Summary of the experiments carried out at EON2003.

2. Interoperability of Protégé-2000 1.8 and WebODE 2.0 [Corcho *et al.*, 2003]. In this experiment, the travel ontology has been developed in WebODE, exported in RDF(S), and imported in Protégé. At this point the ontologies in the two ontology development tools were compared. Finally, the ontology was exported from Protégé to RDF(S), imported to WebODE, and the results were analyzed again.
3. Interoperability of DOE [Isaac *et al.*, 2003] with OilEd, OntoEdit, Protégé-2000, and WebODE. This set of experiments concerned both RDF(S) and OWL, which were used both as interchange language, between those ontology development tools that already supported it (i.e., for OWL, only WebODE and Protégé). Experiments were performed with the traveling domain ontology.
4. Import in SemTalk [Fillies, 2003] the travel ontology modeled in KAON, OilEd, OntoEdit, Protégé, and WebODE using DAML and RDF(S).
5. In another experiment, Protégé was used for modeling the traveling domain ontology and Protégé's plugins were used for exporting the ontology to UML and OWL [Knublauch, 2003].

The RDF(S) Interoperability Benchmarking

The Knowledge Web Network of Excellence organizes several benchmarking initiatives. One of these activities concerns the evaluation of the interoperability of ontology devel-

opment tools using RDF(S) as interchange language²⁸. This initiative started before the benchmarking initiative presented in this thesis and is, at the time of writing this thesis, at the end of the analysis of the *Experiment* phase.

The main characteristics of the interoperability benchmarking using RDF(S) as interchange language are thoroughly described in [García-Castro and Gómez-Pérez, 2005a, García-Castro and Gómez-Pérez, 2005b], and are outlined here.

Methodology. The RDF(S) interoperability benchmarking uses the Knowledge Web benchmarking methodology introduced in Section 2.1.2.

Target. The tools targeted by this benchmarking initiative are ontology development tools, ontology repositories, and other tools that have the ability to export ontologies to, and import ontologies from the RDF(S) language.

Goals. The goals of the RDF(S) Interoperability Benchmarking are the same as those of the OWL Interoperability Benchmarking: to assess and improve the interoperability of ontology development tools using RDF(S) as interchange language, and to identify the fragment of knowledge that these tool can share by means of the RDF(S) language.

Organization of the benchmarking. The benchmarks defined in the RDF(S) interoperability benchmarking are divided in three benchmark suites: an RDF(S) Import Benchmark Suite, an RDF(S) Export Benchmark Suite, and an RDF(S) Interoperability Benchmark Suite. The definition of three suites is due to the necessity of evaluating the importers and exporters of the ontology development tools to the language (RDF(S)) used in the benchmarking activity.

²⁸http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

3. Approach

This chapter presents the approach followed in the development of this thesis. We first list the insufficiencies of the State of the Art, we present the goals that we want to achieve to overcome these insufficiencies, the limitations posed, and the hypothesis under which we are going to develop the work presented in this thesis. Finally, we outline the general guidelines that we will follow in the development of the work.

3.1. Insufficiencies of the State of the Art

The idea of the Semantic Web is to interchange knowledge by means of standard languages and, since the interest for the Semantic Web has raised in the last few years, the development of software applications that support the Semantic Web has increased. Moreover, the W3C Consortium¹ puts a lot of efforts in the definition of standard languages for interoperability.

However, the availability of several tools and languages is not enough to ensure interoperability, as the interchange of ontologies depends also from the importers and exporters of the tools to the language chosen for this interchange. Indeed, how the RDF(S) interoperability benchmarking showed [García-Castro *et al.*, 2006], the level of interoperability among ontology development tools using RDF(S) is not yet satisfactory.

The State of the Art presents insufficiencies with regard to the organization of interoperability benchmarking activities, the availability of benchmark suites, and the software support for benchmarking.

Interoperability benchmarking. In the past, the only activities concerning the interoperability evaluation of Semantic Web technology have been the EON 2003 Workshop and the interoperability benchmarking using RDF(S) as interchange language organized by Knowledge Web. However, there have not yet been benchmarking activities involving interoperability evaluation using OWL, and we do not have any information about the interoperability of the tools.

Benchmark suites. There are other benchmark suites available with the purpose of the evaluation of the interoperability of ontology tools (for example the three suites that compose the RDF(S) Interoperability Benchmark Suite for ontology development tools) but none of these suites is intended for assessing their interoperability using OWL as interchange language.

¹<http://www.w3.org/>

Also the OWL Test Cases might be considered as a benchmark suite for the import of OWL ontologies, but they are not suitable for that, as we explained in Section 2.3.2.

Software support. Although some interoperability evaluations took place in the past, there is no software for supporting in evaluating interoperability or in automatizing the analysis of the results. To the best of the authors' knowledge, the only development in that direction is the case of *rdfsbs* [García-Castro, 2006], a tool developed for the execution of the RDF(S) benchmarking experiments on the WebODE ontology engineering workbench.

3.2. Goals

The goals of the work presented in this thesis provide solutions to two of the insufficiencies of the State of the Art mentioned above, in particular:

Interoperability benchmarking. In this thesis we start with the organization of a benchmarking activity concerning the interoperability of ontology development tools. According to the classifications of benchmarking presented in Section 2.1, our benchmarking activity is **competitive** (i.e., it compares ontology development tools) and focused on **performance** (i.e., it compares the performances of the importers and exporters of ontology development tools).

The main goals of the benchmarking are the following:

- To assess and improve the interoperability of ontology development tools using OWL as interchange language. This would result in knowing the current interoperability of ontology development tools and in maximizing the knowledge that these tools can interchange, while minimizing the information addition or loss.
- To identify the fragment of knowledge that ontology development tools can share using OWL as interchange language. As this fragment becomes larger, more expressive ontologies can be interchanged among ontology development tools.

Benchmark suites. After organizing the benchmarking activity, there was the necessity to define a benchmark suite for evaluating the import of OWL Lite ontologies into ontology development tools. According to the definition of benchmark suite presented in Section 2.1.1, our benchmark suite contains the single benchmarks and several additional information: the tools target of the benchmark suite and their functionalities, the criteria used in the evaluation of the results of the benchmark suite, other material relative to the experiments, and the procedure to execute the experiments.

3.3. Scope

Interoperability benchmarking. The work presented in this thesis instantiates the Knowledge Web benchmarking methodology. In particular, the tasks that have been carried out are all those of the *Plan* phase and the *Experiment definition* task of the *Experiment* phase.

Interoperability scenario. We have described in Section 2.3.1 the four different approaches to the interoperability in the Semantic Web. In this work, we apply the pivot approach, using OWL as interchange language.

The language we have chosen is the less expressive of the three species of the OWL Web Ontology Language, OWL Lite, as this choice allows to extend this work towards OWL DL and OWL Full.

We have also chosen the RDF/XML syntax² for writing OWL ontologies as this is the most used by ontology development tools for serializing OWL ontologies.

Software support. This work does not include the creation of any software that can be used for automatizing the execution of the experiments (like the *rdfsbs* tool mentioned above), the development of such a tool might be a future part of the benchmark suite.

3.4. General vision for approaching the solution

There are two levels on which the work presented in this thesis will be developed, according to the goals we described in section 3.2. The first one is to organize a benchmarking activity, and the second one is to define a benchmark suite for the import of OWL Lite ontologies in ontology development tools to be used in the benchmarking.

Interoperability benchmarking. In order to organize a benchmarking activity, we have instantiated the methodology for benchmarking ontology tools presented in Section 2.1.2.

In the organization of the benchmarking, we studied the technology and planned the benchmarking activity before starting to define a benchmark suite for interoperability. We have used the same approach for the experimentation that was used in the RDF(S) Interoperability Benchmarking. We realized that we needed three benchmark suites: one for import, one for export, and one for interoperability. However, we did not need to define all of them from scratch, since the suites for export and interoperability have been reused from those of the RDF(S) Interoperability Benchmarking [García-Castro, 2006].

Benchmark suite. The process to define the benchmark suites for the evaluation of ontology development tools interoperability using OWL as interchange language started with the study of the knowledge models of OWL and of the ontology development tools,

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

since in the definition of the ontologies that compose the suites, acquaintance with the knowledge models reduces the possibilities of designing redundant or invalid ontologies.

Before starting with the definition of the ontologies used in the benchmarks, we tried two approaches that were not successful, then we found the one that we thought was the correct one. In this approach, we divided the vocabulary terms of OWL into groups and then defined the ontologies. Each ontology has been described in different ways, for a better understandability.

4. Benchmarking OWL interoperability

This section presents the instantiation of the Knowledge Web benchmarking methodology (described in Section 2) for benchmarking the interoperability of ontology development tools using OWL as interchange language.

The benchmarking methodology developed in Knowledge Web was used in the RDF(S) Interoperability Benchmarking and has been adopted also for the OWL Interoperability Benchmarking presented in this thesis. We introduce here how the tasks of the methodology have been carried out. At the time of writing this thesis, we do not have yet started with the *Experiment* phase, hence we will not present the instantiation of the *Improve* phase.

4.1. Plan Phase

The tasks in this phase are those related with the organization and the planning of the benchmarking activity. In the case of the OWL interoperability benchmarking, they are:

Goals identification. The benchmarking initiator wanted to know, and to improve if needed, the interoperability of WebODE¹, the ontology development tool developed in his group (the Ontology Engineering Group, OEG²), comparing it with other ontology development tools and decided to start a benchmarking activity. The benchmarking initiator identified the benefits (e.g., an improved tool brings compliance with current technology and a broader user base) and estimated the costs related to the benchmarking process.

Tools and metrics identification. The functionalities that would be investigated in the benchmarking activity are the importers and exporters of ontology development tools to OWL. The benchmarking initiator also decided the best objective criteria to evaluate these functionalities, which are the fragment of OWL that the tools can interchange correctly and the OWL components that WebODE can interchange with other ontology development tools.

Participant identification. The benchmarking initiator selected the members of the OEG group that did take part in the benchmarking process, forming the benchmarking

¹<http://webode.dia.fi.upm.es/WebODEWeb/index.html>

²<http://www.oeg-upm.net>

team after obtaining their availability. The team is composed by the benchmarking initiator, the current maintainer of WebODE, and the developer of the OWL importers and exporters of WebODE.

Proposal writing. The benchmarking initiator wrote the proposal document as a web page³, which is publicly available for comments, proposals, suggestions, and feedback in general. This page provides all information about the benchmarking, in particular:

- The goals of the benchmarking (i.e., to identify the fragment of the knowledge models shared by the tools that they can use to interoperate and to improve the interoperability of ontology development tools using OWL as interchange language).
- The benefits for developers and users of ontology development tools (i.e., to know in detail the actual interoperability of their ontology development tools, to know the set of terms in which interoperability between their ontology development tools can be achieved, and to show that their ontology development tools are able to interoperate and are among the best).
- The benefits for the Semantic Web community and the industrial sector (i.e., to obtain a significant improvement in the interoperability of ontology development tools, to know the best practices that are performed when developing the import and export facilities of ontology development tools, to obtain instruments to assess the interoperability of ontology development tools, and to know the best-in-class ontology development tools regarding interoperability).
- The experiments (i.e., the different stages that compose the benchmarking execution, that will be presented in detail in Section 4.2 below).
- The benchmark suites used in this benchmarking activity: OWL Lite Import Benchmark Suite, OWL Export Benchmark Suite, and OWL Interoperability Benchmark Suite. For each of them, an additional web page has been set up, with all the benchmarks, the procedure to execute the experiments, the evaluation criteria, and the templates that must be filled with the results of the experiments.

Management involvement. The proposal document was submitted to the director of the OEG group for her approval. She discussed the proposal with the benchmarking team, gave her authorization to proceed with the benchmarking activity, and granted the financial support to the benchmarking team.

Partner selection. The benchmarking team collected information about the ontology development tools developed by partners in the Knowledge Web project, then extended its research to other ontology development tools developed by external organizations. For each tool found, different types of information were collected and reported: its name, the

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

name of the organization that develops it, and the home page of the tool. Table 4.1 shows these informations for the tools found by the benchmarking initiator.

Tool	Organization	URL
Cerebra Workbench	Cerebra	http://www.cerebra.com/products/repos_bench_it.html
COBrA	University of Edinburgh	http://xspan.org/applications/cobra/
InferEd	Intellidimension	http://www.intellidimension.com/pages/site/products/infered/
Integrated Ontology Development Toolkit	IBM	http://www.alphaworks.ibm.com/tech/semanticstk
IODE	Ontologyworks	http://www.ontologyworks.com/iode.php
IsaViz	W3C Consortium	http://www.w3.org/2001/11/IsaViz/
LinkFactory Workbench	Language and computing nv	http://www.landcglobal.com/pages/linkfactory.php
KAON2	Universität Karlsruhe	http://kaon2.semanticweb.org/
OilEd	University of Manchester	http://oiled.man.ac.uk/
OntoStudio	Ontoprise	http://www.ontostudio.de/
Protégé	Stanford University	http://protege.stanford.edu/
SemanticWorks	Altova	http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor.html
SemTalk	Semtation	http://www.semtalk.com/
SMORE	Mindswap	http://www.mindswap.org/2005/SMORE/
SWOOP	Mindswap	http://www.mindswap.org/2004/SWOOP/
TERMINAE	University Paris North	http://www-lipn.univ-paris13.fr/%7Eszulman/TERMINAE.html
WebODE	Universidad Politécnica de Madrid	http://webode.dia.fi.upm.es/WebODEWeb/index.html

Table 4.1.: Ontology development tools capable of importing/exporting OWL

The functionalities of these tools were investigated, in order to find which of them can import and export OWL ontologies. Besides their functionalities, the ontology development tools considered should also meet the criteria previously defined by the benchmarking team in order to take part to the benchmarking process: the importance of the tool in the community or in the industry, the use of the latest available technologies, the wide use of the tool by the community, and the public availability of the tool.

A call for participation was sent to the members of Knowledge Web and some of the interested organizations provided useful feedback. The document containing the benchmarking proposal was therefore modified according to their suggestions on the process to follow in the benchmarking process. The updated proposal document was then approved by all partners.

Planning and resource allocation. The directors of all the organizations and all the benchmarking teams discussed the updated benchmarking proposal and planned the future stages of the benchmarking process, taking into account the available resources, both financial and human, and ended up with a tentative scheduling of the tasks of the next phases: deadlines for definition of the benchmarking, proposal of working days for the execution of the experiments, and for the presentation of the results.

4.2. Experiment Phase

The execution of the experiments has not yet started by the time of writing this thesis. We describe here how the *Experiment definition* task was carried out.

Experiment definition. Before starting with the definitions of the ontologies contained in the benchmarks, the benchmarking teams studied various technical documents of the W3C consortium about the OWL Web Ontology language, especially the OWL Guide [Smith *et al.*, 2004], the OWL Test Cases [Carroll and Roo, 2004], and the OWL Abstract Syntax and Semantics [Patel-Schneider *et al.*, 2004], in order to increase its knowledge about the language used in the benchmarking. The first result was to create an interoperability benchmark suite for OWL Lite, as this is the less expressive of the sublanguages of OWL and benchmark suites for OWL DL and OWL full might be created on top of the one for OWL Lite.

The benchmarking teams decided to follow the same approach of the RDF(S) interoperability benchmarking, assessing the functionalities of the importers and exporters of ontology development tools before actually evaluating their interoperability.

For this purpose, we needed three different benchmark suites: two of them are intended to evaluate importers and exporters of the ontology development tools, while the third one is intended for the evaluation of interoperability of ontology development tools. These three benchmark suites were assessed and agreed on by all the members involved in the benchmarking initiative, to ensure their quality.

The benchmarking teams also realized that the RDF(S) Export and Interoperability Benchmark Suites [García-Castro, 2006] could be used in the OWL Interoperability benchmarking. The Export and the Interoperability suites, indeed, are independent from the interchange language. Therefore, these two suites have been reused without any modifications, but changing their names to OWL Export Benchmark Suite and OWL Interoperability Benchmark Suite respectively.

The benchmarking teams set up a planning for the execution of the experiments, which is shown in Figure 4.1 and consists of three stages, described below.

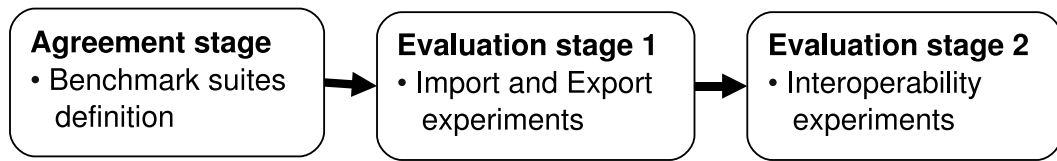


Figure 4.1.: The experimentation phases

Agreement stage. The agreement stage concludes all the theoretical work that has been produced for the definition of the benchmark suites. All the partners involved in the definition of the benchmark suites shall agree on their quality. This agreement is necessary to state that the benchmark suites satisfy the principles defined in Section 2.1 and to allow the benchmarking teams to start the experiments on the ontology development tools with the same benchmarks.

Evaluation stage 1. In the first stage of evaluation, the benchmarking teams execute the import and export experiments on the tools. The outcomes of the tools are used to fill a result template, which may contain also comments and other information, for a more complete overview of each experiment (see also Section 5.6 for more details). These results are then analyzed: they show strengths and weaknesses of the importers and exporters of the tool and provide guidelines on how to improve them.

Evaluation stage 2. In the second stage of the evaluation, experiments on the interoperability of the ontology development tools are carried out. The result of these experiments will allow for reaching the goals of the benchmarking initiative: it will become clear which fragment of knowledge ontology development tools can share and it will be possible to assess the current state of interoperability among ontology development tools and to provide guidelines for their improvement.

All the material produced (i.e., the three benchmark suites and all additional data and information necessary to the execution of the experiments) is made available to the partners on the benchmarking web page⁴. This gives also the possibility to people or organizations, which are interested in benchmarking OWL interoperability, but that have not been contacted, to participate or to perform their own experiments.

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

5. The OWL Lite Import Benchmark Suite

In this chapter we present in detail the process followed to develop the OWL Lite Import Benchmark Suite, which is intended to evaluate the OWL import capabilities of ontology development tools. We show here the approaches followed: the first and the second were discarded, whereas the third is the one which revealed the most convenient for our case. This last approach foresees the creation of groups of benchmarks, according to the OWL Lite Semantics and Abstract Syntax ([Patel-Schneider *et al.*, 2004]). For each of this groups, we present the OWL axioms involved, some example of the benchmarking defined, the criteria that shall be used for the evaluation of the outcomes of the ontology development tools, and the procedure that has to be followed for carrying out the experiments.

5.1. Hypothesis

The assumptions we took for the development of the OWL Interoperability Benchmark Suite are the following:

- The use of OWL Lite for the definition of the ontologies.
- To use the RDF/XML syntax for writing OWL ontologies as this is the most used by ontology development tools for importing and exporting ontologies.
- To only define correct ontologies. The ontologies defined in the benchmarks do not contain syntactic or semantic errors, and in order to ensure the syntactic correctness of the ontologies, we decided to use an OWL validator. Our choice was the WonderWeb OWL Ontology Validator¹.
- To only define simple ontologies. In the list of benchmarks that we have defined there are no complex ontologies, but only simple ones. Indeed, when importing a complex ontology, it becomes difficult to detect errors, for example when some components are lost in the import process.
- The number of benchmarks should be small. Benchmarking is a process that consumes a lot of resources, and any increase in the number of benchmarks leads to an increment in the time required for performing the experiments and for the subsequent analysis of the results.

¹<http://phoebus.cs.man.ac.uk:9999/OWL/Validator>

5.2. Typology of benchmarks

One of the goals of this benchmark suite is to identify the fragment of the OWL knowledge model that ontology development tools can use to interchange ontologies. We decided to concentrate our efforts in the definition of benchmarks that cover the largest possible fragment of the knowledge model of OWL, but we realized that we had to consider also the different variants for expressing knowledge allowed by the RDF/XML syntax in which the OWL ontologies could be written

Hence, we decided to separate the benchmarks related with the OWL Lite knowledge model (i.e., defining benchmarks to check the import of the different combinations of the OWL Lite vocabulary terms) from those that are related with the variants of the RDF/XML syntax, by defining a separated group.

5.3. Benchmarks that depend on the OWL Lite knowledge model.

These benchmarks check the import of ontologies with different combinations of the OWL Lite vocabulary terms.

Before starting with the definition of the single benchmarks, we analyzed the OWL vocabulary and tried to figure out the possible development process for the composition of the suite, but our first and second tentatives revealed ineffective. We briefly describe here these two approaches, and then extensively present the final one.

5.3.1. The first approach

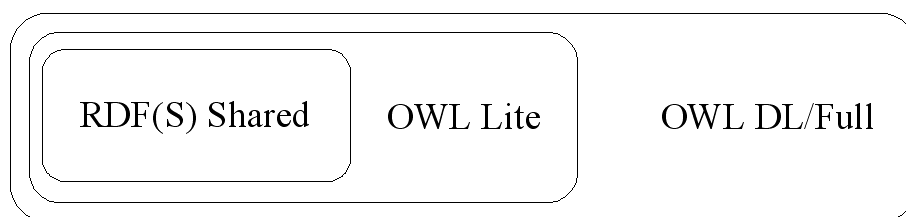


Figure 5.1.: The first approach to build the OWL Lite Import Benchmark Suite

The OWL language uses also some of RDF and RDF Schema vocabulary terms, so our idea was to split the OWL and RDF(S) vocabularies in three groups, like Figure 5.1 shows, and obtain a layered benchmark suite that could be executed incrementally.

However, the vocabulary terms of RDF(S) cannot form a stand-alone group, since they must be used in conjunction with the OWL vocabulary. As an example, consider that the terms `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:range`, and `rdfs:domain` cannot be used without being related to `owl:DatatypeProperty`, `owl:ObjectProperty`, or `owl:Class`. Moreover, the class `rdfs:Literal` is

used for expressing the range of datatype properties and therefore must be used in conjunction with the `owl:DatatypeProperty` vocabulary term.

Hence, we discarded this solution, keeping only the idea of a layered approach to the solution, as the semantics of OWL Lite, OWL DL and OWL Full suggests.

5.3.2. The second approach

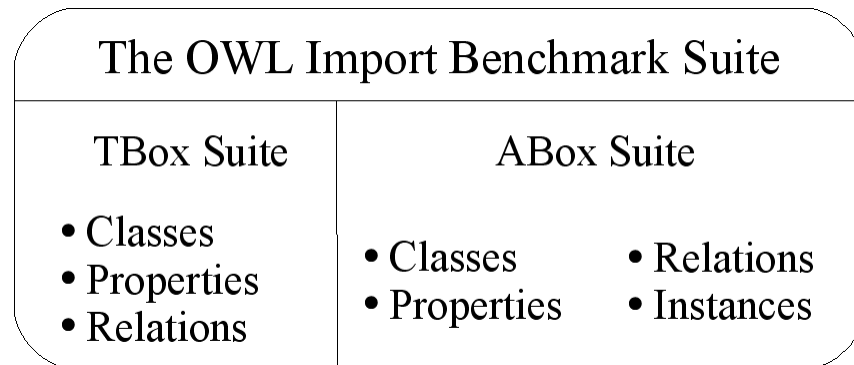


Figure 5.2.: The second approach to build the OWL Lite Import Benchmark Suite

The second idea for defining the benchmark suite considered the creation of two distinct benchmark suites: one for TBox with definitions of classes, properties, and relations among them and another for ABox adding instances to at least some of the TBox ontologies. With this separation, we wanted to resemble the semantics that OWL inherits from the Description Logics formalism. However, this idea was discarded for several reasons: the final benchmark suite would have comprised too many benchmarks, most of them would have been very similar to each other or redundant, and the time and human resources needed to carry out the experiments and to analyze the results would have been too long.

5.3.3. The final approach

We finally decided to simplify the second tentative and split the benchmarks related to the OWL knowledge model into three parts, each of them focusing on one of the main components of the OWL knowledge model: classes, properties, and instances.

Each part was further divided into groups, for the sake of clarity. The groups have been defined by keeping in mind the productions of the OWL Semantics and Abstract Syntax [Patel-Schneider *et al.*, 2004], although we did not take into consideration these productions for defining the groups of the OWL Lite Import Benchmark Suite, but only for defining the ontologies.

Indeed, there is some difference between the groups of productions defined by the OWL Abstract Syntax and the groups defined in the OWL Lite Import Benchmark Suite. The productions of the OWL Abstract syntax are divided in three groups (i.e., ontology, facts,

and axioms), while we have created more groups in order to have a clearer organization of the benchmarks.

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

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

We considered at most one or two OWL vocabulary terms at a time, then we considered all the possible combinations of these terms with the remaining. We then defined in natural language the ontologies that would have composed the groups. Moreover, according to the goals of benchmarks, we decided to discard those vocabulary terms that do not contribute to the OWL expressiveness, which are those for annotation properties, versioning, and heading.

When the number of the ontologies defined was large, we pruned the benchmark suite and also decided to consider the combinations of the OWL vocabulary terms with a cardinality of zero, one, and two, assuming that the result for higher cardinality equals the result for cardinality two.

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

The conventions used in the productions are: a start symbol of the language is capitalized, otherwise it is lowercase. Terminals are quoted. Alternatives are separated by a colon (|) or given in different productions. Square brackets ([. . .]) indicate elements that occur at most once. Braces ({. . .}) indicate elements that can occur any number of times, including zero.

Benchmarks for Classes

The initial step was to enumerate the different possibilities that can be used for the description of classes in OWL Lite and list the corresponding OWL vocabulary.

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

We decided to group the benchmarks according to the following criteria: description of classes and class hierarchies, description of class equivalence and description of classes by set operator.


```
restriction(myNs:hasCar someValuesFrom(myNs:Car))
)
```

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

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

Group B: Class equivalence

These benchmarks are intended for checking the import of class equivalences. This group includes classes equivalent to value restrictions and cardinality restrictions on properties and classes equivalent to intersection of classes. Moreover, this group and group A are also intended to stress the ability of ontology development tools to cope with the difference between a subclass relation and an equivalent class relation. The benchmarks in this group are alike those in Group A that define hierarchies: the difference is that in Group A there are primitive classes (i.e., with modality = 'partial') and in group B there are defined classes (i.e., with modality = 'complete').

The vocabulary terms involved in this group are:

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

The productions we used for defining the benchmarks are:

```
axiom ::= 'Class('classID modality
          {super}')'
axiom ::= 'EquivalentClasses('classID classID {classID}')'
modality ::= 'complete'
super ::= classID | restriction | description
restriction ::= 'restriction('datavaluedPropertyID
                dataRestrictionComponent')'
            | 'restriction('individualvaluedPropertyID
                individualRestrictionComponent')'
dataRestrictionComponent ::= 'allValuesFrom('dataRange')'
                          | 'someValuesFrom('dataRange')'
                          | cardinality
```

```

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

```

Group C: Class defined with set operators

The ontologies defined in this group consist of classes defined by means of 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.

The vocabulary terms involved in this group are :

```
owl:intersectionOf, owl:Class
```

The production we used for defining these benchmarks are:

```

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

```

Benchmarks for Properties

We identified the different ways provided by the OWL Abstract Syntax to define a property. In OWL Lite, properties can be either object properties, i.e., properties that link a class with another class, or datatype properties, i.e., properties that link a class to a data value (e.g., a string, an integer, a boolean value and so on).

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

Group D: Property and property hierarchies

These benchmarks are intended for checking the import of properties and property hierarchies.

The vocabulary terms involved are:

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

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

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

Group E: Property with domain and range

The ontologies in this group describe properties that have from one to three domain and/or range constraints. We do not consider in this group properties with no range and domain constraint, as they are included in group D.

The vocabulary terms involved in this group 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

These benchmarks are intended for checking the import of equivalences among object properties and among datatype properties and the import of object properties which are one inverse of the other. It is not possible to define the inverse of a datatype property, since the inverse relation would have a literal (i.e., a data value) as its domain, and this is not allowed in OWL Lite.

The vocabulary terms involved are:

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

In this group we use the following axioms:

```

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

dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference

```

Group G: Global cardinality constraints and logical characteristics of properties

In OWL, object and datatype properties can be further described with more expressive characteristics. The ontologies in this group describe properties with domain and range, that 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.

The vocabulary terms involved are:

```

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

```

The axiom used for generating ontologies in this group are:

```

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

dataRange ::= datatypeID | 'rdfs:Literal'
datatypeID ::= URIreference
classID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference

```

Benchmarks for Instances

We enumerate the possibilities that the OWL Abstract syntax allows for defining facts about instances (individuals). In OWL Lite, individuals are instances of classes related by properties to other individuals. There are also special built-in properties for asserting relationships among them. Individuals can be either named or anonymous.

Group H: Single individuals

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

The only vocabulary terms involved in this group are `owl:Class` and `rdf:type`. The OWL Lite axioms used in this group are:

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

Group I: Named individual and properties

Individuals can be related one to each other by means of user defined properties. In each ontology of this group, there is one object or datatype property with possibly different classes as domain and range and individuals as instance of these classes. Moreover, the object and datatype properties are simple (i.e., there are no logical characteristics of properties specified) and, in the case of datatype properties, there are also data values (we used only strings).

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

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

The axioms covered in this group are:

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

```

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

```

Group J: Anonymous individuals and properties

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

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

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

The OWL Lite axioms involved 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 K: Individual identity

The OWL vocabulary consists also of built in predicates (i.e., terms) to express basic relations among individuals. This terms can be used to state that two individuals are the

same or that are different and to state that in a set of individuals, each of them is different from the others.

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

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

The axioms involved in this group are:

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

The reader can note that there is no explicit production that generates the vocabulary terms `owl:AllDifferent` and `owl:distinctMembers`. The abstract syntax of OWL allows to produce only pairwise disjoint individuals, and these two vocabulary terms are indeed intended as a shortcut for expressing that, given a set of individuals, each of them is unique and different from all the other in the set.

5.4. Benchmarks that depend on the syntax.

These benchmarks check the correct import of OWL ontologies with the different variants of the RDF/XML syntax, as stated in the RDF/XML specification. These benchmarks are arranged into different categories, each of which checks one different aspect of the possible RDF/XML variants.

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

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

- using an absolute URI references, e.g.,

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

- using an URI reference relative to a base URI, e.g.,

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

- using an URI reference transformed from `rdf:ID` attribute values, e.g.,

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

- using an URI reference relative to an *ENTITY* declaration, e.g.,

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

Abbreviations. Each statement in OWL is composed by exactly one subject, one predicate and one object and this is also what happens in the RDF/XML syntax. However, there are cases in which the RDF/XML syntax relaxes this constraint, allowing to group statements with a same subject or to shorten the RDF/XML code, like the examples in the following four subcategories show. Here there are benchmarks for evaluating the import of descriptions of empty nodes, of multiple properties, of typed nodes, of string literals, and of blank nodes. For each subcategory we have defined two benchmarks.

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

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

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

- **Multiple properties or properties with multiple ranges and domains.** The following example shows how to group statements relative to a same property.

```
<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 benchmark is intended for evaluating the correct interpretation of 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>
```

5.5. Description of the benchmarks

Each benchmark of the OWL Lite Import Benchmark Suite, as Table 5.1 shows, is described by:

Identifier	ISG03
Description	Import a single functional object property with domain a class and range another class
Formal Description	$\top \sqsubseteq \forall hasHusband$ $\top \sqsubseteq \forall hasHusband^- . Woman$ $\top \sqsubseteq \forall hasHusband . Man$
Graphical representation	
RDF/XML file	<pre>... <owl:Ontology rdf:about="#" > <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 5.1.: The description of a benchmark of the OWL Lite Import Benchmark Suite.

- A unique **identifier** (i.e., *ISG03*, where *IS* denotes the OWL Import Benchmark Suite, *G* is the group to which the benchmark belongs to, and *01* is an increasing number).
- A **description** in natural language of the ontology defined in the benchmark.
- A **formal description** in the Description Logics notation of the ontology defined in the benchmark.
- A **graphical representation** of the ontology.

- A **file** with the ontology in the RDF/XML syntax. All the files have been syntactically validated against the WonderWeb OWL Ontology Validator³.

The resulting OWL Lite Import Benchmark Suite is available in a public web page⁴ and is composed of 82 benchmarks that are classified in 12 groups, each identified by one letter (from **A** to **L**), to manage them easily. The list of all the benchmarks that compose the OWL Lite Import Benchmark Suite can be found in Appendix A; the OWL files have not been included here, but they can be found in the OWL Lite Import Benchmark Suite web page.

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

5.6. Evaluation criteria

The evaluation criteria of the OWL Import Benchmark Suite are defined as follows:

Modelling (YES/NO). This criterion concerns the ability of the ontology development tool to model the components of the ontology described in the benchmark. *Modelling* is YES if the tool can model the component and NO otherwise.

Execution (OK/FAIL). If the execution of the benchmark is normally carried out without any problem, and the tool always produces the expected result, then the *Execution* is OK, otherwise it is FAIL. When a failed execution occurs, the benchmarking participants are asked to provide information about this failure for obtaining the practices used when developing the OWL importers. The information required is the following:

- The reasons for the ontology development tool to fail in the execution of the benchmark.
- In case the ontology development tool is repeating the benchmark, because in a previous execution it failed, the changes implemented in the tool.

Information added or lost. The information added or lost in the import of the ontology when executing the benchmark.

Table 5.2 shows the different possible results that a benchmark execution can produce. As it can be seen, there is no *Right* or *Wrong* result, as the benchmarking is not intended to provide a ranking of the tools. Furthermore, any combination of results can be possible as they depend on the decisions taken by each tool developer when implementing the behaviour of the tool when it has to deal with components outside of its knowledge model.

The only pattern that can be identified in the results is that when a tool does import an ontology with a component that does not belong to its knowledge model, there will always be some information loss (at least the component that the tool cannot model).

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

⁴http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html

Modelling	Execution	Information added	Information lost
YES	OK	Some	Some
YES	OK	Some	None
YES	OK	None	Some
YES	OK	None	None
YES	FAIL	Some	Some
YES	FAIL	Some	None
YES	FAIL	None	Some
YES	FAIL	None	None
NO	OK	Some	Some
NO	OK	None	Some
NO	FAIL	Some	Some
NO	FAIL	None	Some

Table 5.2.: The different results of a benchmark execution

For example, consider the benchmark ISG03, which requires the import of a functional property. The developer of an ontology development tool that does not support functional properties, has two possibilities to implement the importer of the tool when dealing with a functional property. The first is to completely ignore the property and discard it: in that case the information lost is the functional property and its domain and range, if present. The second one is to import only the property as not functional, in that case the information lost is only the "functional" characteristic of the property.

Table 5.3 shows the results of a fictitious experimentation involving five ontology development tools, identified by A, B, C, D, and E, which execute benchmark ISG03 (*Import a single functional object property with domain a class and range another class*). For each tool we define the expected result and provide the result of *Modelling*, *Execution*, *Information added*, and *Information lost*.

In this example, tools A and B can model functional object properties and therefore their *Modelling* result is *YES*, whereas tools C, D, and E cannot model functional object properties and therefore their *Modelling* result is *NO*.

Tool A. The expected result is to import an ontology containing the functional object property *hasHusband* with class *Woman* as domain and class *Man* as range. All these components are imported, hence the result of *Execution* is *OK*. A `rdfs:label` is added to all of the resources imported, and this is reported as *Information added*.

Tool B. The expected result is to import an ontology containing a functional object property with a class as domain and another class as range. Tool B imports the functional object property, the two classes and the range relation, but not the domain relation, i.e., the property *hasHusband* has the class *Man* as range, but not the class *Woman* as its domain. Since this does not coincide with the expected result, the *Execution* result is *FAIL*. The domain constraint that has not been imported has to be reported as *Information lost*.

Tool	ID	Expected result	Modelling	Execution	Information added	Information lost
A	ISG03	A functional object property with a class as domain and another class as range	YES	OK	A label in all the components	-
B	ISG03	A functional object property with a class as domain and another class as range	YES	FAIL	-	The property's domain
C	ISG03	An object property with domain a class and domain another class	NO	OK	-	-
D	ISG03	An object property with domain a class and domain another class	NO	OK	A cardinality of 1 in the property	The property as functional
E	ISG03	An object property with domain a class and domain another class	NO	FAIL	-	The property

Table 5.3.: Fictitious results of executing benchmark ISG03

Tool C. The expected result is to obtain the object property *hasHusband* with domain class *Woman* and range class *Man*. The imported ontology is the same as the expected result, therefore the *Execution* is *OK*. As no additional knowledge is added by the tool in the import process, nothing has to be reported as *Information added* or *Information lost*.

Tool D. The expected result is to obtain the object property *hasHusband* with range class *Man* and domain class *Woman*. The imported ontology contains the expected result, therefore the *Execution* is *OK*. However, during the import process, the tool added a cardinality of 1 to the object property, since its internal knowledge model requires every property to have a cardinality. This knowledge added has to be reported as *Information added*.

Tool E. The expected result is to obtain the object property *hasHusband* with class *Man* as its range and class *Woman* as its domain. However, Tool E imports the two classes but it does not import at all the functional object property, although its expected result is to import it as an object property. The *Execution* result of Tool E is therefore *FAIL*, and it loses all the information about the property (i.e., its "functionality" , as expected, but also the range and domain constraints) when importing the ontology, and this has to be reported as *Information lost*.

5.7. Procedure for executing the benchmark suite

In order to compare the outcomes of the ontology development tools and to guarantee that all the participants in the benchmarking activities execute the experiments in the same conditions, we defined a procedure for carrying out the experiments, that must be followed by each participant. This procedure consists of the sequence of steps described below and illustrated with an example.

Step 1. To define the expected result, which is the result that the ontology development tool is supposed to produce after importing the ontology contained in the benchmark. The expected result is defined either by modelling in the ontology development tool the ontology expected from importing the file with the OWL ontology, or by defining it informally (i.e., in natural language).

Step 2. To import, from the ontology development tool, the file which contains the OWL ontology defined in the benchmark.

Step 3. After the benchmark has been executed, to check whether the expected ontology modelled equals the imported one: if they are not the same, the differences should be reported along with the reason of the two ontologies being different.

Furthermore, it should be identified whether some information has been added or was lost in the import process.

We illustrate this procedure with an example. Consider two ontology development tools (tool X and tool Y), and execute on them the benchmark ISF03 (*Import an object property with domain a class and range another class, which is inverse of another object property*), which consists of an ontology containing two object properties:

i *hasParent*, with class *Child* as domain and class *Person* as range

ii *hasChild*, which is the inverse of *hasParent*.

The execution of the benchmark consists of the following steps.

Step 1. Suppose that tool X can model all the components of the ontology. In this case the expected result is "To obtain the two properties *hasParent* and *hasChild*, the inverse relation between them and the domain (*Child*) and range (*Person*) of *hasParent*".

Suppose now that tool Y can model all the components of the ontology and that automatically adds a label to all imported resources. In this case the expected result is: "To obtain the two properties *hasParent* and *hasChild*, the inverse relation between them, the domain (*Child*) and range (*Person*) of *hasParent* and a label in each resource".

Step 2. To open the file *ISF03.owl* containing the ontology within tool X and tool Y.

Step 3. Suppose that tool X imports all the components of the ontology, but not the inverse relation: in this case the *Information lost* is the inverse relationship between the properties.

Suppose that tool Y correctly imports all the components of the ontology and adds a label to all resources: in this case something like "the ontology development tool adds to every resource an `rdfs:label` with the name of the resource" must be reported as *Information added*.

5.8. Automation of the execution of the benchmark suite

The procedure required to execute all the benchmarks of the OWL Lite Import Benchmark Suite is time consuming, as the user must, for each ontology, define the expected result, load the ontology, compare the results and fill the necessary fields in the documentation template.

However, it is possible to automate this procedure, or at least some part of it, thus saving time in the benchmark suite execution. For example, it might be possible to develop a software application that relies on the API of the ontology development tool for importing or comparing the ontologies, leaving to the user only the tasks of defining the expected result and compile the documentation.

As an example of such a software application, consider the *rdfsbs* Java application (see [García-Castro, 2006]), which was developed right after the definition of the RDF(S)

interoperability benchmark suites. It is an application that relies on the WebODE API and allows to automate, in that ontology development tool, the following tasks:

1. To import files containing RDF(S) ontologies written in the RDF/XML syntax.
2. To create a text file containing the description of an ontology, which has been modelled or imported into WebODE.
3. To create ontologies in WebODE and exporting them in RDF(S) format to files.

It is evident that the *rdfsbs* application, although tailored for the RDF(S) benchmarking in the WebODE ontology development tool, can easily be adapted to the OWL Lite Import Benchmark Suite, since WebODE supports the import and export of OWL ontologies.

The steps of the procedure to execute the OWL Lite Import Benchmark Suite, that can be automatized using the API of WebODE as follows:

Step 1. To define the expected results in WebODE.

Step 2. To import each ontology of the OWL Lite Import Benchmark Suite into WebODE.

Step 3. To export text files with the description of the expected and of the imported ontology to files, to ease the comparison of the ontologies.

The automatization of the procedure can be implemented not only for WebODE, but also for every ontology development tool that has an API.

It is straightforward to see the great reduction of efforts required for carrying out the experiments: the only effort required to execute the whole OWL Lite Import Benchmark Suite would be to compare the original benchmarks with the texts files produced by *rdfsbs*.

Moreover, as many ontology development tools have their own API, the development of software applications like *rdfsbs* would not be difficult. This would result in a twofold improvement:

- On the one side, as already said, the time and efforts needed for the execution of the benchmark suites would evidently decrease, leaving more time for the analysis of the results.
- On the other side, it would be possible, if everybody would be able to automatize the procedure for the execution of the experiments, to carry out more benchmarks in the same time needed for a manual execution of the experiments. This would imply also the possibility to define more comprehensive and detailed benchmark suites.

5.9. Towards import benchmark suites for OWL DL and OWL Full

We have seen in Section 2.2.1 that OWL is composed of three layered sublanguages. Hence, the OWL Lite Import Benchmark Suite described in this paper, although just dealing with the OWL Lite sublanguage, might be used also for the evaluation of the importers from OWL DL and OWL Full of ontology development tools.

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

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

OWL DL

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

The OWL Import Benchmark Suite provides us with benchmarks to evaluate the different combinations of the OWL Lite vocabulary terms and the different variants of the RDF/XML syntax.

As Figure 5.3 shows, to cover the OWL DL sublanguage of OWL, we should need also to consider:

- The different combinations of the OWL Lite vocabulary terms according to their use in OWL DL, as OWL DL imposes less restrictions to their use. Table 5.9 shows the differences in the use restrictions of the vocabulary terms for OWL Lite and DL⁵.
- The different combinations of the OWL DL vocabulary terms not allowed in OWL Lite, between themselves and between the OWL Lite vocabulary terms. The vocabulary terms that are allowed in OWL DL and not allowed in OWL Lite are: `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf`, `owl:hasValue`, and `owl:DataRange`.

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

- A class identifier. These benchmarks are already defined for OWL Lite.

⁵<http://www.w3.org/TR/owl-ref/#Sublanguages-def>

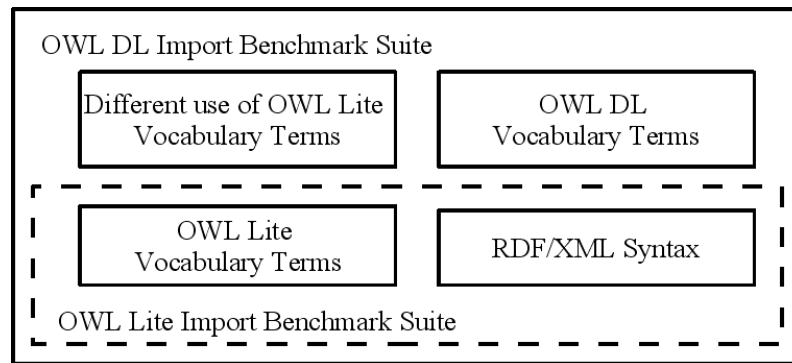


Figure 5.3.: The OWL DL Import Benchmark Suite

- An exhaustive enumeration of individuals. These benchmarks are not defined for OWL Lite.
- Property restrictions with value and cardinality constraints. Benchmarks are defined for OWL Lite considering restrictions in the object of the properties with 0 and 1 cardinality constraints. New benchmarks should be defined for cardinalities in the object of the properties greater than 1 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 of the OWL Lite Import Benchmark Suite could be reused without any modification and, therefore, any ontology development tool that already performed the experiments of the OWL Import Benchmark Suite presented in section 5, would not need to repeat them.

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

OWL Full

OWL Full has the same the vocabulary terms than OWL DL, but 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 also to augment the meaning of both vocabularies.

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

- All the RDF(S) vocabulary can be used within OWL Full.

Vocabulary Terms	OWL Lite restrictions	OWL DL restrictions
owl:cardinality owl:minCardinality owl:maxCardinality	Object must be 0 or 1	Object must be any integer ≥ 0
owl:equivalentClass rdfs:subClassOf	Subject must be class names	No restriction
owl:equivalentClass rdfs:subClassOf rdf:type	Object must be class names or restrictions	No restriction
rdfs:domain	Object must be class names	No restriction
owl:allValuesFrom owl:someValuesFrom rdfs:range	Object must be class names or datatype names	No restriction
owl:intersectionOf	Used only with lists of length greater than one of class names or restrictions	No restriction

Table 5.4.: Restrictions in the use of OWL Lite and OWL DL

- OWL Full has no separation between classes, datatypes, datatype properties, object properties, annotation properties, individuals, data values, and the built-in vocabulary.
- Axioms in OWL Full must not be well formed.

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

This import benchmark suite for OWL Full should consider all the possible combinations of the OWL and RDF(s) vocabularies terms and, as the number of these combinations is huge, it would be necessary to prune the generation of benchmarks like was done for the RDF(S) Import Benchmark Suite [García-Castro and Gómez-Pérez, 2005b].

6. Conclusions and future work

There are several lessons we learned during the development of this thesis, concerning both the evolution of the benchmarking process and the design and definition of the benchmark suites.

6.1. Lesson learned

6.1.1. Benchmarking process

A benchmarking initiative is a long and complex process, composed by several interdependent tasks. Hence, starting such an initiative requires to have in mind a clear idea about the whole benchmarking process. The main issues to consider and to take care of are on the one side planning and time (i.e., time needed for defining the suites, for executing the experiments, for analyzing the results, and so on) and on the other the human resources required during the process (i.e., how many people are involved and in which task).

Planning and time. The planning of the whole benchmarking process must be both strict and flexible. This apparent contradiction can be explained as follows.

We already stressed the fact that benchmarking is a time consuming process, therefore it is mandatory, before the beginning, to define a time line as early as possible, and to try to keep the work to be done in the limits set. It is also necessary that such a time line foresees possible delays in the commitment of the work, due to unforeseeable reasons, as happened during the development of this work.

When we started the development of the benchmark suites, we realized that we had to set up, beside a benchmarking suite for interoperability, also two more benchmark suites for evaluating the importers and exporters of the ontology development tools. This implied that we needed more time for the experiment definition, but also that we would have had to split the experiments execution task in two stages, like it is described in Section 4. It also implied that the time required for the analysis of the results of the experimentation would have been considerably longer.

Human Resources. Another important point to take into account when undertaking a benchmarking initiative is represented by the human resources on which to count on. It is straightforward to affirm, that the more people play an active role in the benchmarking, the less time is needed in each cycle of the benchmarking process. Hence, it is important to think about possible defection of people during the process and being able to replace them.

When deciding how many tools to use in the benchmarking, it is necessary to consider that executing manually all the experiments of the three suites requires a lot of time to each participant, and also the participants are prone to commit errors. It is possible to spare time and avoid unwanted errors by automatizing the experiment execution. For a discussion on that, refer to Section 5.8 and to Section 6.1.2 below.

Hence, the benchmark initiator must define a strict planning of time and human resources from the beginning, but he/she has to take into account the possibility to have to reallocate both resources on the fly at any point of the development.

6.1.2. Benchmark suite

During the development of the benchmark suites, we have learned some useful lessons from several situations in which we had to take a decision for continuing our work.

The most important point to consider is that the creation of a benchmark suite is a task that has a trade-off between the necessity of performing a thorough evaluation activity and the time needed for executing all the benchmarks and for analyzing the results.

The definition of a benchmark suite for the interchange of ontologies using a layered language like OWL is a source of additional difficulties, as there is not a unique language to build a benchmark suite for.

Finally, it is also important to select the appropriate level of expressiveness and complexity of the ontologies that compose the benchmark suite, as discussed in Section 5.2.

Time. Time is an issue in both the definition of the single ontologies, in their validation, and also in ensuring that they can cover the largest possible fragment of knowledge model of OWL. However, these issues can be hardly overcome, as they require for the largest part an human active involvement.

Time is also an issue when executing the benchmarking. However, in this case, as explained in Section 5.8, it is possible to spare time by programming some simple software application, that automatically carries out all the experiments. This allows to decrease the time needed for the execution of the whole benchmark suite to a few minutes and reserve the spared time for the analysis of the results. Moreover, if the automatization were done for all ontology development tools, then it would be possible to define benchmark suite that contain a higher number of benchmarks.

The OWL language. The use of a language like OWL, composed by three layered languages, posed the first important question in the development of this work: which of them to use? There can be two feasible approaches: define some benchmarks for each of the OWL species or focus on the simpler sublanguage and then build benchmark suites for the other sublanguages on top of that one.

Our choice was to start from the less expressive of OWL sublanguages (OWL Lite), as it is more manageable and the one with the smallest knowledge model. This permits to cover a significant part of OWL Lite's knowledge model with a reasonable number of benchmarks.

Simple ontologies. The definition of simple ontologies helps a quick identification of possible problems and/or inefficiencies in the importers and exporters of the ontology development tools, as discussed in Section 5.2. Moreover, as the vocabulary terms of OWL Lite are the same than those of OWL DL and OWL Full, the definition of simple ontologies helps to reuse the benchmark suites in either extending the OWL Lite Import Benchmark Suite towards OWL DL and OWL Full or extending the OWL Export Benchmark Suite by broaden the shared knowledge model defined by the most common features of the ontology development tools.

6.2. Future work

The work presented in this thesis can be extended in several directions. We present here those that can be carried out in the next future or that will be executed within the Knowledge Web Network of Excellence.

Automating the execution of the experiments. The manual execution of the experiments is a very time consuming task. To this end, an automatization of this task would be of great help.

Since most of the ontology development tools are released with an API to access their functionalities, the development of an application that takes care of executing the benchmarks should be quite straightforward. An example of such an application is *rdfsbs* [García-Castro, 2006], a Java program for executing the benchmark suites in the RDF(S) interoperability benchmarking.

Completing the first iteration of the benchmarking process. The most immediate extension to the work presented in this thesis is to carry out the import and export experiments on the ontology development tools. After this it is possible to move to the interoperability experiments and to their analysis.

The result obtained by performing the experiments will help the developers of the ontology development tools to improve their tools.

Moving to OWL DL and OWL Full. The OWL Lite Import Benchmark Suite for OWL Lite can be extended to cover OWL DL: this is described in detail in Section 5.9. This extension is quite straightforward, as it only requires to add benchmarks for the OWL vocabulary terms whose use is forbidden in OWL Lite and with combinations of these terms with the other.

More complex is the case of extending the OWL Lite Import Benchmark Suite to OWL Full. The semantics of OWL Full allows to combine every vocabulary term of OWL and RDF(S) without any limitations, like RDF(S) allows. Hence, a thorough import benchmark suite for OWL Full would require to develop, on top of the suite for OWL DL, a new benchmark suite that covers the differences between OWL DL and OWL Full. The main characteristic of this additional suite is that its development should follow the guidelines described in [García-Castro and Gómez-Pérez, 2006]: consider all the possible

combinations of the OWL and RDF(s) vocabularies terms and, as the number of these combinations is high, prune the generation of benchmark suite.

Bibliography

- [Ahmed and Rafiq, 1998] P.K. Ahmed and M. Rafiq. Integrated benchmarking: an holistic examination of select techniques for benchmarking analysis. *Benchmarking for Quality Management and Technology*, 5(3):225–242, 1998.
- [Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [Benslimane *et al.*, 2005] D. Benslimane, S. Suwanmanee, and P. Thiran. OWL-Based Approach for Semantic Interoperability. In IEEE, editor, *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA 05)*, pages 145–150, 2005.
- [Bicer *et al.*, 2005] Veli Bicer, Asuman Dogac, Ozgur Kilic, and Gokce B. Laleci. Archetype-based semantic interoperability of web service messages in the health care domain. *International Journal on Semantic Web and Information Systems*, 1(4):1–22, October-December 2005.
- [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.
- [Bray *et al.*, 2000] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, 2000. W3C Recommendation 6 October 2000.
- [Bull *et al.*, 1999] J. M. Bull, R. A. Davey, D. S. Henty, L. A. Smith, and M. D. Westhead. A methodology for benchmarking java grande applications. In *Proceedings of the ACM 1999 conference on Java Grande*, pages 81–88, 1999.
- [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 O. Chorcho and Y. Sure, editors, *EON 2003 Evaluation of Ontology-based Tools*, volume 87. CEUR, 2003.
- [Camp, 1989] R. C. Camp. *Benchmarking: The Search for Industry Best Practices that Lead to Superior Performance*. Quality Resources, 1989.
- [Carroll and Roo, 2004] J.J. Carroll and J. De Roo. OWL Web Ontology Language test cases. Technical report, W3C, February 2004.

- [Corcho and Sure, 2003] O. Corcho and Y. Sure, editors. *2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, volume 87 of *CEUR-WS*, 2003.
- [Corcho *et al.*, 2003] 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 O. Corcho and Y. Sure, editors, *EON 2003 Evaluation of Ontology-based Tools*, volume 87. CEUR, 2003.
- [Damiani *et al.*, 2005] E. Damiani, S. David, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati. Open world reasoning in the semantic web: a preliminary study. In P. Bouquet and G. Tummarello, editors, *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop*, volume 166. CEUR, december 2005.
- [Donini *et al.*, 1996] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996.
- [Euzenat and Stuckenschmidt, 2003] J. Euzenat and H. Stuckenschmidt. The family of language approach to semantic interoperability. In B. Omelayenko and M. Klein, editors, *Knowledge transformation for the semantic web*, pages 49–63, Amsterdam (NL), 2003. IOS press.
- [Fikes and Kehler, 1985] Richard Fikes and Tom Kehler. The role of frame-based representation in reasoning. *Commun. ACM*, 28(9):904–920, 1985.
- [Fillies, 2003] Christian Fillies. SemTalk EON2003 Semantic Web Export / Import Interface Test. In O. Corcho and Y. Sure, editors, *EON 2003 Evaluation of Ontology-based Tools*, volume 87. CEUR, 2003.
- [García-Castro and Gómez-Pérez, 2005a] R. García-Castro and A. Gómez-Pérez. Guidelines for benchmarking the performance of ontology management APIs. In *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, 2006] R. García-Castro and A. Gómez-Pérez. Benchmark suites for improving the RDF(S) importers and exporters of ontology development tools. In York Sure and John Domingue, editors, *Proceedings of the European Semantic Web Conference 2006*, LNCS, pages 155–169. Springer-Verlag, 2006.

- [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.*, 2005] García-Castro *et al.* Specification of a methodology, general criteria, and benchmark suites for benchmarking ontology tools. Deliverable D2.1.4, KnowledgeWeb EU-IST Network of Excellence IST-2004-507482, February 2005.
- [García-Castro, 2006] R. García-Castro. Prototypes of tools and benchmark suites for benchmarking ontology building tools. Deliverable D2.1.5, KnowledgeWeb EU-IST Network of Excellence IST-2004-507482, January 2006.
- [Gómez-Pérez *et al.*, 2003] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Springer Verlag, 2003.
- [Gruber, 1993] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [Hendler and Herman, 2004] J. Hendler and I. Herman. Web Ontology Language OWL, 2004. <http://www.w3.org/2004/OWL/>.
- [Horrocks and Pan, 2003] I. Horrocks and J. Z. Pan. RDFS(FA): A DL-ised Sublanguage of RDFS. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, 2003.
- [Horrocks and Patel-Schneider, 2004] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.
- [Horst, 2005] H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3(2):59, 2005.
- [IEEE, 1990] IEEE. IEEE Std 610-1990 IEEE Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossary, 1990. New York, NY.
- [IEEE, 1991] IEEE. IEEE Standard Glossary of Software Engineering, February 1991.
- [IEEE, 1998] IEEE. Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology, 1998.
- [Isaac *et al.*, 2003] Antoine Isaac, Véronique Malaisé, and Raphaël Troncy. Using XSLT for Interoperability: DOE and The Travelling Domain Experiment. In O. Chorcho and Y. Sure, editors, *EON 2003 Evaluation of Ontology-based Tools*, volume 87. CEUR, 2003.
- [Kifer *et al.*, 1990] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. Technical Report TR-90-003, 1, 1990.

- [Knublauch, 2003] Holger Knublauch. Case Study: Using Protege to Convert the Travel Ontology to UML and OWL. In O. Chorcho and Y. Sure, editors, *EON 2003 Evaluation of Ontology-based Tools*, volume 87. CEUR, 2003.
- [Lankford, 2000] W. M. Lankford. Benchmarking: Understanding the Basics. *Coastal Business Journal*, 1(1), 2000.
- [Patel-Schneider *et al.*, 2004] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, 2004. <http://www.w3.org/TR/owl-semantics/>.
- [Rector *et al.*, 2004] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In E. Motta and N. Shadbolt, editors, *Proceedings of the European Conference on Knowledge Acquisition*, pages 63–81. Springer-Verlag, 2004.
- [Shirazi *et al.*, 1999] B. Shirazi, L.R. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, and E. Huh. Dynbench: A dynamic benchmark suite for distributed real-time systems. In *Proceedings of the 11 IPPS/SPDP'99 Workshops*, pages 1335–1349. Springer-Verlag, 1999.
- [Sill, 1996] D. Sill. comp.benchmark Frequently Asked Questions, 1996. version 1.0.
- [Sim *et al.*, 2003] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 74–83, Portland, OR, 2003.
- [Smith *et al.*, 2004] M. K. Smith, C. Welty, D. L. McGuinness, and (eds.). OWL Web Ontology Language Guide. W3C recommendation 10 february 2004. Technical report, W3C, February 2004.
- [Sole and Bist, 2005] T. D. Sole and G. Bist. Benchmarking in technical information. *IEEE Transactions on Professional Communication*, 38(2):77–82, June 2005.
- [Spendolini, 1992] M. J. Spendolini. *The Benchmarking Book*. AMACOM, New York, 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, 22-27 June 2003.
- [T.Berners-Lee and Masinter, 1998] R. Fielding T.Berners-Lee and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. Technical report, IETF, 1998.
- [Volz, 2004] Rapahel Volz. *Web ontology reasoning with logic databases*. PhD thesis, AIFB Karlsruhe, 2004.

[Wache *et al.*, 2004] H. Wache, L. Serafini, and R. García-Castro. Survey of scalability techniques for reasoning with ontologies. Deliverable D2.1.1, KnowledgeWeb EU-IST Network of Excellence IST-2004-507482, August 2004.

[Wiremann, 2003] T. Wiremann. *Benchmarking Best Practices in Maintenance Management*. Industrial Press, 2003.

A. List of benchmarks in the OWL Lite Import Benchmark Suite

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

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

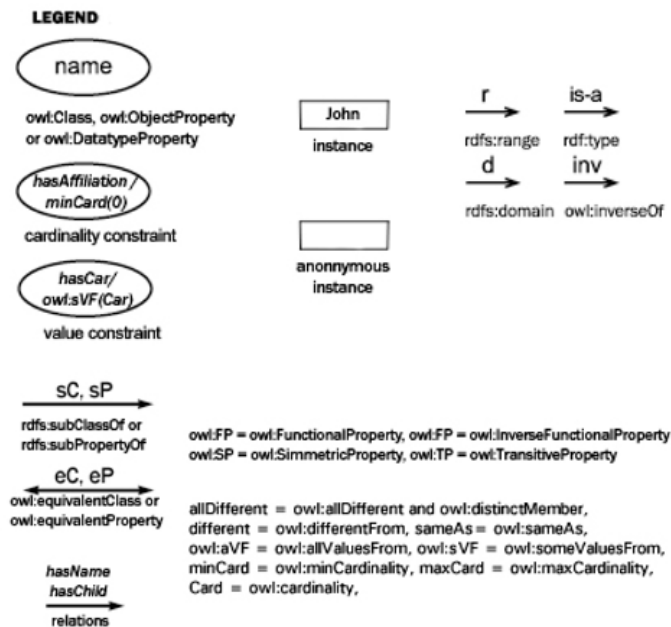
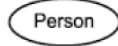

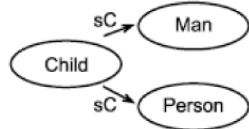
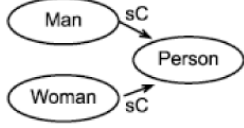


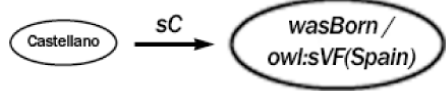
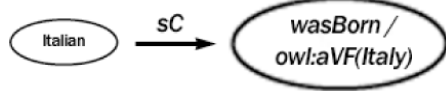
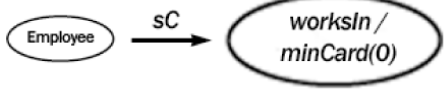
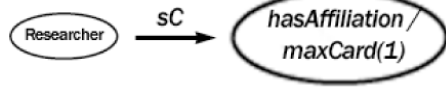
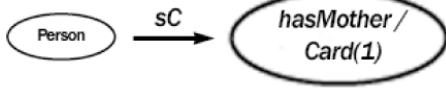


Figure A.1.: Notation used in the benchmarks of the OWL Lite Import Benchmark Suite

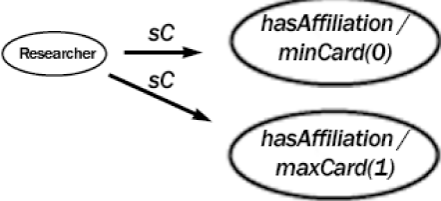
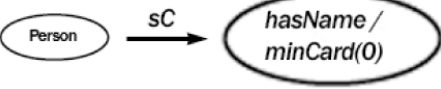
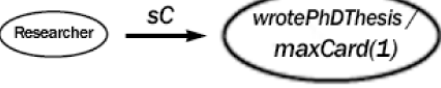
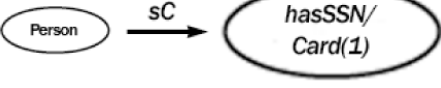

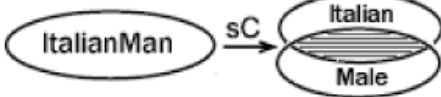
Class benchmarks

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	
ISA04	Import several classes subclass of a single class	
ISA05	Import two classes, each subclass of the other	
ISA06	Import a class, subclass of itself	
ISA07	Import a class which is subclass of an anonymous class defined by an owl:someValuesFrom value constraint in an object property	
ISA08	Import a class which is subclass of an anonymous class defined by an owl:allValuesFrom value constraint in an object property	
ISA09	Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property	
ISA10	Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property	
ISA11	Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property	

(continued on next page)

(continued from previous page)

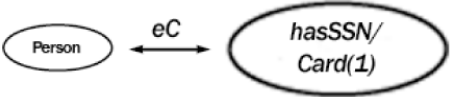
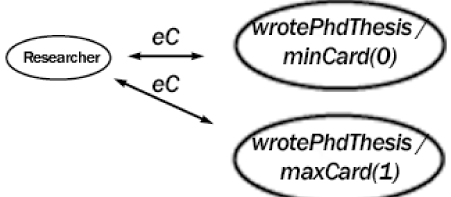
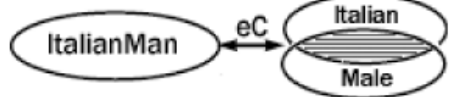
<p>ISA12</p>	<p>Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in an object property</p>	
<p>ISA13</p>	<p>Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property</p>	
<p>ISA14</p>	<p>Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property</p>	
<p>ISA15</p>	<p>Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property</p>	
<p>ISA16</p>	<p>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</p>	
<p>ISA17</p>	<p>Import a class which is subclass of an anonymous class defined by the intersection of two other classes</p>	

Group B: Class equivalences



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

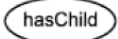

ISB10	Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property	
ISB11	Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 and an owl:maxCardinality=1 cardinality constraints in a datatype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

Group C: Classes defined with set operators

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

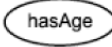
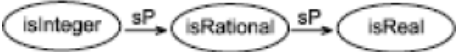
Property benchmarks

Group D: Property hierarchies

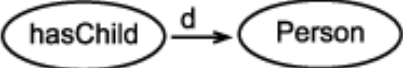
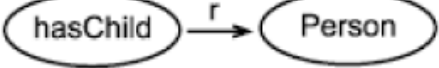
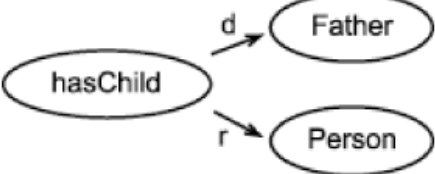
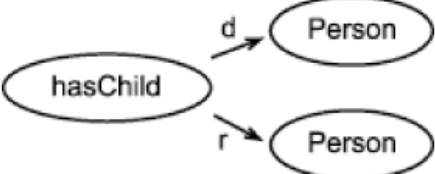
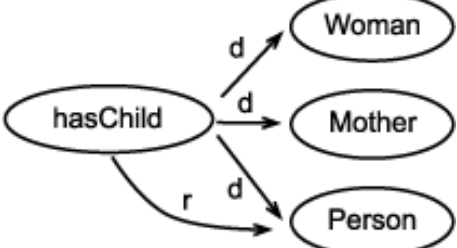
ID	Description	Graphical representation
ISD01	Import a single object property	
ISD02	Import an object property that is subproperty of another object property that is subproperty of a third one	

(continued on next page)

(continued from previous page)

ISD03	Import a single datatype property	
ISD04	Import a datatype property that is subproperty of another datatype property that is subproperty of a third one	

Group E: Properties with domain and range

ID	Description	Graphical representation
ISE01	Import a single object property with domain a class	
ISE02	Import a single object property with range a class	
ISE03	Import a single object property with domain a class and range another class	
ISE04	Import a single object property with domain and range the same class	
ISE05	Import a single object property with domain multiple classes and range a class	

(continued on next page)

(continued from previous page)

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

Group F: Relations between properties

ID	Description	Graphical representation
ISF01	Import several object properties with domain a class and range another class, which are all of them equivalent	

(continued on next page)

(continued from previous page)

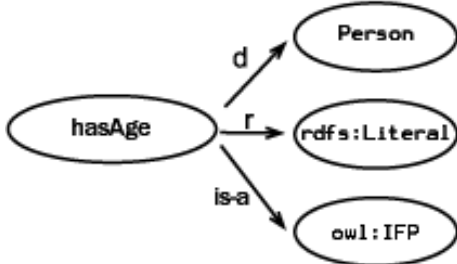
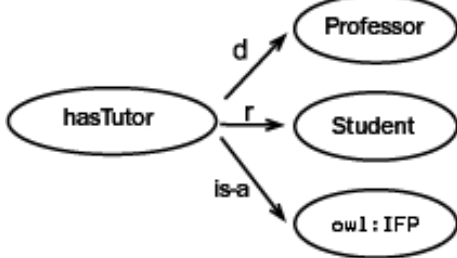
ISF02	Import several datatype properties with domain a class and range <code>rdfs:Literal</code> , which are all of them equivalent	
ISF03	Import an object property with domain a class and range another class, which is inverse of another object property	

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	

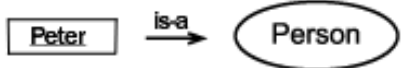
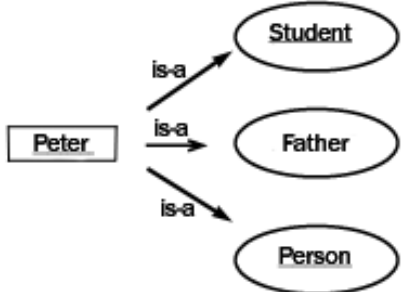
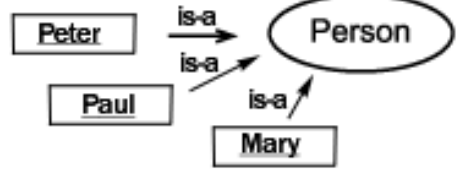
(continued on next page)

(continued from previous page)

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	
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 rdfs:Literal, 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 rdfs:Literal, and one individual of the class that has the datatype property with several literals	

Group J: Anonymous individuals and properties

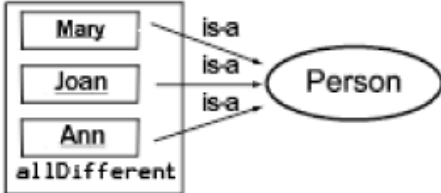
ID	Description	Graphical representation
ISJ01	Import one class, one object property with domain and range the class, and one anonymous individual of the class that has the object property with another individual of the same class	
ISJ02	Import two classes, one object property with domain one class and 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 rdfs:Literal, 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	

(continued on next page)

(continued from previous page)

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

B. Description of the ontologies in the Description Logics formalism.

This appendix provides a formal description of the ontologies that compose the OWL Lite Import Benchmark Suite¹, in the Description Logics formalism.

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

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

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

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

¹http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/import.html

the Description Logics formalism.

ID	<p style="text-align: center;">Here there is the description in natural language...</p> <div style="border: 1px solid black; padding: 2px; text-align: center;">...and here the one in the Description Logics formalism.</div>
ISE06	<p style="text-align: center;">Import a single object property with domain a class and range multiple classes</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> $\top \sqsubseteq \forall hasChild^-.Person$ $\top \sqsubseteq \forall hasChild.Person$ $\top \sqsubseteq \forall hasChild.Human$ $\top \sqsubseteq \forall hasChild.Child$ </div>

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

Class benchmarks

Group A: Class hierarchies

ISA01	<p style="text-align: center;">Import a single class</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">Person</div>
ISA02	<p style="text-align: center;">Import a single class, subclass of a second class which is subclass of a third one</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">Child \sqsubseteq Man \sqsubseteq Person</div>
ISA03	<p style="text-align: center;">Import a class that is subclass of two classes</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Child \sqsubseteq Man Child \sqsubseteq Person </div>
ISA04	<p style="text-align: center;">Import several classes subclass of a single class</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Woman \sqsubseteq Person Man \sqsubseteq Person </div>
ISA05	<p style="text-align: center;">Import two classes, each subclass of the other</p> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Male \sqsubseteq Man Man \sqsubseteq Male </div>
ISA06	<p style="text-align: center;">Import a class, subclass of itself</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">Woman \sqsubseteq Woman</div>
ISA07	<p style="text-align: center;">Import a class which is subclass of an anonymous class defined by an owl: someValuesFrom value constraint in an object property</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">Driver \sqsubseteq $\exists hasCar.Car$</div>
ISA08	<p style="text-align: center;">Import a class which is subclass of an anonymous class defined by an owl: allValuesFrom value constraint in an object property</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">Italian \sqsubseteq $\forall wasBorn.Italy$</div>

(continued on next page)

(continued from previous page)

ISA09	<p>Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in an object property</p> <p style="text-align: center;">Employee $\sqsubseteq \geq 0$ worksIn</p>
ISA10	<p>Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property</p> <p style="text-align: center;">Researcher $\sqsubseteq \leq 1$ hasAffiliation</p>
ISA11	<p>Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property</p> <p style="text-align: center;">Man $\sqsubseteq = 1$ hasMother</p>
ISA12	<p>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</p> <p style="text-align: center;">Researcher $\sqsubseteq \geq 0$ hasAffiliation Researcher $\sqsubseteq \leq 1$ hasAffiliation</p>
ISA13	<p>Import a class which is subclass of an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property</p> <p style="text-align: center;">Person $\sqsubseteq \geq 0$ hasName</p>
ISA14	<p>Import a class which is subclass of an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property</p> <p style="text-align: center;">Researcher $\sqsubseteq \leq 1$ wrotePhDThesis</p>
ISA15	<p>Import a class which is subclass of an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property</p> <p style="text-align: center;">Person $\sqsubseteq = 1$ hasSSN</p>
ISA16	<p>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</p> <p style="text-align: center;">Researcher $\sqsubseteq \geq 0$ wrotePhDThesis Researcher $\sqsubseteq \leq 1$ wrotePhDThesis</p>
ISA17	<p>Import a class which is subclass of a class defined by the intersection of two other classes</p> <p style="text-align: center;">ItalianMan $\sqsubseteq (\text{Italian} \sqcap \text{Male})$</p>

Group B: Class Equivalences

ISB01	<p>Import several classes which are all of them equivalent</p> <p style="text-align: center;">Italian \equiv Italiano \equiv Italiana</p>
ISB02	<p>Import a class which is equivalent to an anonymous class defined by an owl:someValuesFrom value constraint in an object property</p> <p style="text-align: center;">Driver $\equiv \exists hasCar.Car$</p>
ISB03	<p>Import a class which is equivalent to an anonymous class defined by an owl:allValuesFrom value constraint in an object property</p> <p style="text-align: center;">Italian $\equiv \forall wasBorn.Italy$</p>
ISB04	<p>Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=1 cardinality constraint in an object property</p> <p style="text-align: center;">Employee $\equiv \geq 1 worksIn$</p>
ISB05	<p>Import a class which is equivalent to an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in an object property</p> <p style="text-align: center;">Researcher $\equiv \leq 1 hasAffiliation$</p>
ISB06	<p>Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in an object property</p> <p style="text-align: center;">Man $\equiv = 1 hasMother$</p>
ISB07	<p>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</p> <p style="text-align: center;">Researcher $\equiv (\leq 1 hasAffiliation \sqcap \geq 0 hasAffiliation)$</p>
ISB08	<p>Import a class which is equivalent to an anonymous class defined by an owl:minCardinality=0 cardinality constraint in a datatype property</p> <p style="text-align: center;">Person $\equiv \geq 0 hasName$</p>
ISB09	<p>Import a class which is equivalent to an anonymous class defined by an owl:maxCardinality=1 cardinality constraint in a datatype property</p> <p style="text-align: center;">Researcher $\equiv \leq 1 wrotePhDThesis$</p>

(continued on next page)

(continued from previous page)

ISB10	<p>Import a class which is equivalent to an anonymous class defined by an owl:cardinality=1 cardinality constraint in a datatype property</p> <p style="text-align: center;">Person \equiv 1hasSSN</p>
ISB11	<p>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</p> <p style="text-align: center;">Researcher \equiv ≥ 0 wrotePhDThesis Researcher \equiv ≤ 1 wrotePhDThesis</p>
ISB12	<p>Import a class which is equivalent to an anonymous class defined by the intersection of two other classes</p> <p style="text-align: center;">ItalianMan \equiv (Italian \sqcap Male)</p>

Group C: Class defined by set operators

ISC01	<p>Import a class which is intersection of two other classes</p> <p style="text-align: center;">ItalianMan \equiv (Italian \sqcap Male)</p>
ISC02	<p>Import a class which is intersection of several other classes</p> <p style="text-align: center;">ItalianMan \equiv (Italian \sqcap Male \sqcap Person)</p>

Property benchmarks

Group D: Property hierarchies

ISD01	<p>Import a single object property</p> <p style="text-align: center;"><i>hasChild</i></p>
ISD02	<p>Import an object property that is subproperty of another object property that is subproperty of a third one</p> <p style="text-align: center;"><i>isFatherOf</i> \sqsubseteq <i>isGrandFatherOf</i> \sqsubseteq <i>isAncestorOf</i></p>
ISD03	<p>Import a single datatype property</p> <p style="text-align: center;"><i>hasAge</i></p>
ISD04	<p>Import a datatype property that is subproperty of another datatype property that is subproperty of a third one</p> <p style="text-align: center;"><i>isInteger</i> \sqsubseteq <i>isRational</i> \sqsubseteq <i>isReal</i></p>

Group E: Properties with domain and range

ISE01	<p>Import a single object property with domain a class</p> $\top \sqsubseteq \forall hasChild^- . Person$
ISE02	<p>Import a single object property with range a class</p> $\top \sqsubseteq \forall hasChild . Person$
ISE03	<p>Import a single object property with domain a class and range another class</p> $\top \sqsubseteq \forall hasChild^- . Father$ $\top \sqsubseteq \forall hasChild . Person$
ISE04	<p>Import a single object property with domain and range the same class</p> $\top \sqsubseteq \forall hasChild^- . Person$ $\top \sqsubseteq \forall hasChild . Person$
ISE05	<p>Import a single object property with domain multiple classes and range a class</p> $\top \sqsubseteq \forall hasChild^- . Mother$ $\top \sqsubseteq \forall hasChild^- . Woman$ $\top \sqsubseteq \forall hasChild^- . Person$ $\top \sqsubseteq \forall hasChild . Person$
ISE06	<p>Import a single object property with domain a class and range multiple classes</p> $\top \sqsubseteq \forall hasChild^- . Person$ $\top \sqsubseteq \forall hasChild . Person$ $\top \sqsubseteq \forall hasChild . Human$ $\top \sqsubseteq \forall hasChild . Child$
ISE07	<p>Import a single datatype property with domain a class</p> $\top \sqsubseteq \forall hasSSN^- . Person$
ISE08	<p>Import a single datatype property with range <code>rdfs:Literal</code></p> $\top \sqsubseteq \forall hasName . rdfs:Literal$
ISE09	<p>Import a single datatype property with domain a class and range <code>rdfs:Literal</code></p> $\top \sqsubseteq \forall hasName^- . Person$ $\top \sqsubseteq \forall hasName . rdfs:Literal$
ISE10	<p>Import a single datatype property with domain multiple classes and range <code>rdfs:Literal</code></p> $\top \sqsubseteq \forall hasChildNamed^- . Mother$ $\top \sqsubseteq \forall hasChildNamed^- . Woman$ $\top \sqsubseteq \forall hasChildNamed . rdfs:Literal$

Group F: Property equivalences

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

Group G: Logical characteristics of properties

ISG01	<p>Import a single transitive object property with domain and range the same class</p> $\text{hasFriend}^+ \sqsubseteq \text{hasFriend}$ $\top \sqsubseteq \forall \text{ hasFriend}^- . \text{Person}$ $\top \sqsubseteq \forall \text{ hasFriend} . \text{Person}$
ISG02	<p>Import a single symmetric object property with domain and range the same class</p> $\text{hasFriend} \equiv \text{hasFriend}^-$ $\top \sqsubseteq \forall \text{ hasFriend}^- . \text{Person}$ $\top \sqsubseteq \forall \text{ hasFriend} . \text{Person}$
ISG03	<p>Import a single functional object property with domain a class and range another class</p> $\top \sqsubseteq \forall \text{ hasHusband}$ $\top \sqsubseteq \forall \text{ hasHusband}^- . \text{Woman}$ $\top \sqsubseteq \forall \text{ hasHusband} . \text{Man}$
ISG04	<p>Import a single functional datatype property with domain a class and range <code>rdfs:Literal</code></p> $\top \sqsubseteq \forall \text{ hasAge}$ $\top \sqsubseteq \forall \text{ hasAge}^- . \text{Person}$ $\top \sqsubseteq \forall \text{ hasAge} . \text{rdfs:Literal}$

(continued on next page)

(continued from previous page)

ISG05	Import a single inverse functional object property with domain a class and range another class
	$\top \sqsubseteq \forall hasTutor^-$ $\top \sqsubseteq \forall hasTutor^-.Professor$ $\top \sqsubseteq \forall hasTutor^-.Student$

Individual benchmarks

Group H: Single individuals

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)

(continued on next page)

(continued from previous page)

<p>ISI03</p>	<p>Import two classes, one object property with domain one class and range the other class, and one individual of one class that has the object property with an individual of the other class</p> <pre style="border: 1px solid black; padding: 5px;"> T ⊆ ∀hasChild⁻.Mother T ⊆ ∀hasChild.Child Mother(MARY) Child(PAUL) hasChild(MARY, PAUL)</pre>
<p>ISI04</p>	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one individual of the class that has the datatype property with a literal</p> <pre style="border: 1px solid black; padding: 5px;"> T ⊆ ∀hasName⁻.Person T ⊆ ∀hasName.rdfs:Literal Person(MARYSMITH) hasName(MARYSMITH, "Mary")</pre>
<p>ISI05</p>	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one individual of the class that has the datatype property with several literals</p> <pre style="border: 1px solid black; padding: 5px;"> T ⊆ ∀hasName⁻.Person T ⊆ ∀hasName.rdfs:Literal Person(MARYANN) hasName(MARYANN, "Mary") hasName(MARYANN, "Ann")</pre>

Group J: Anonymous individuals and properties

<p>ISJ01</p>	<p>Import one class, one object property with domain and range the class, and one anonymous individual of the class that has the object property with another individual of the same class</p> <pre style="border: 1px solid black; padding: 5px;"> T ⊆ ∀hasChild⁻.Person T ⊆ ∀hasChild.Person Person(JOHN) hasChild(ANON^a, JOHN)</pre> <hr style="width: 20%; margin-left: 0;"/> <p>^aThis denotes an <i>anonymous individual</i></p>
---------------------	--

(continued on next page)

(continued from previous page)

ISJ02	<p>Import two classes, one object property with domain one class and range the other class, and one anonymous individual of one class that has the datatype property with an individual of the other class</p> $\top \sqsubseteq \forall hasChild^-.Parent$ $\top \sqsubseteq \forall hasChild.Person$ $Person(JOHN)$ $hasChild(ANON, JOHN)$
ISJ03	<p>Import one class, one datatype property with domain the class and range <code>rdfs:Literal</code>, and one anonymous individual of the class that has the datatype property with a literal</p> $\top \sqsubseteq \forall hasName^-.Person$ $\top \sqsubseteq \forall hasName.rdfs:Literal$ $hasName(ANON, "Peter")$

Group K: Individual identity

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