A keyword-driven approach for generating OWL DL conformance test data

Raúl García-Castro , Asunción Gómez-Pérez

A B S T R A C T

The conformance of semantic technologies has to be systematically evaluated to measure and verify the real adherence of these technologies to the Semantic Web standards. Current evaluations of semantic technology conformance are not exhaustive enough and do not directly cover user requirements and use scenarios, which raises the need for a simple, extensible and parameterizable method to generate test data for such evaluations. To address this need, this paper presents a keyword-driven approach for generating ontology language conformance test data that can be used to evaluate semantic technologies, details the definition of a test suite for evaluating OWL DL conformance using this approach, and describes the use and extension of this test suite during the evaluation of some tools.

1. Introduction

The W3C Semantic Web Activity has produced different standards[1] that enable technology interoperability in the open environment of the (Semantic) Web. However, a systematic evaluation of the conformance of semantic technologies is required to measure and verify their real adherence to these standards.

Conformance is a primary requirement for semantic technologies and its evaluation essentially covers two scenarios in terms of: (a) *tool validation*, which is mainly relevant to tool developers and involves checking whether the tool correctly meets the specifications and (b) *feature analysis*, which is mainly relevant to tool users and involves checking which parts of the specification the tool covers, either the whole specification or a subset of it.

Current evaluations of semantic technology conformance are not exhaustive enough, both in terms of technology coverage and of standard coverage. However, while other characteristics of semantic technologies (e.g., efficiency or usability) are non-critical in most use scenarios (i.e., users can come to terms with a variation in tool quality), users expect full conformance to the specifications included in standards or to the subset required by them.

Clearly, full conformance evaluation is impossible, since it is not possible to define every possible variation of the requirements included in a certain specification (e.g., to define every possible OWL ontology or SPARQL query), and we need to produce effective conformance evaluation methods and evaluation data. A similar issue is largely covered in the area of software testing where it is acknowledged that, besides expertise in the functionality to be tested and in the domain of the data to be used, effective testing requires understanding the different use scenarios of the software (Burnstein, 2003), in our case, the different semantic technology use scenarios.

The ideal approach would be to cover these use scenarios when evaluating semantic technology conformance, but currently semantic technology users are passive actors in conformance evaluations. First, current conformance evaluations are generic and do not directly cover user requirements and use scenarios and, second, it is difficult for users to evaluate technology conformance on their own, since this is a resource-consuming task and they do not have enough expertise in semantic technologies and their specifications.

This raises the need for a method to define conformance test data that is simple, to facilitate users the definition of test data suited to their use scenarios and the understanding of existing evaluations, and extensible and parameterizable, to allow defining test data as exhaustive as needed.

Previous work on the evaluation of semantic technology conformance with regards to the ontology language model has covered the definition of test data to be used in conformance evaluations and of methods for the execution of these evaluations, both manually and automatically. Once we have a way of automatically executing conformance evaluations, we can afford to increase the exhaustiveness of these evaluations and to involve users in them, that is, to generate larger quantities of test data and to allow users to define these data according to their needs.

This paper presents a keyword-driven approach for generating ontology language conformance test data that can be used to evaluate semantic technologies. The paper only covers the generation

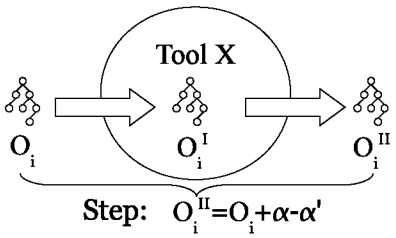

Fig. 1. Steps of a conformance test execution.

of test data and not the use of these data in evaluations, since this is already covered in previous work. Nevertheless, we also describe how we have defined, using this approach, a test suite for evaluating the OWL DL conformance of semantic technologies and present how we have used and extended this test suite during the evaluation of some tools.

This paper is structured as follows. Section 2 presents our understanding of conformance and previous work related to this topic. Section 3 gives an overview of the keyword-driven test suite generation process and, then, Sections 4–6 provide some insights into the definition of the Keyword Library used in this process, the structure of test suite definition scripts, and the implementation of the test suite generator, respectively. Section 7 shows how the OWL DL Import Test Suite was defined following the abovementioned process and Section 8 explains how we have used this test suite to evaluate some tools. Finally, Section 9 includes some discussion about the work presented and Section 10 presents some conclusions and future lines of work.

2. Related work

The conformance characteristic for semantic technologies is related to the ability of these technologies to adhere to existing specifications; in this case, the most relevant specifications are those of the existing ontology representation languages (i.e., RDF(S), OWL and OWL 2).

With respect to an ontology language specification there are several aspects of semantic technology conformance, since such conformance can be evaluated in terms of:

- *The ontology language model.* Since different tools have different internal knowledge representation models, it is important to know the similarities and differences between these internal models and the knowledge representation model of the ontology language.
- *The ontology language serialization.* Existing ontology languages have different serializations, both normative and non-normative (e.g., N3, RDF/XML, OWL/XML, and Turtle). A tool that supports an ontology language should also support at least one of such serializations, including their syntactic variants.
- *The ontology language semantics.* Ontology language specifications include one or more formal semantics that can be used with the ontology language. A tool implementing one of these formal semantics should be consistent with it.

The work presented in this paper only covers conformance regarding the ontology language model and does not cover other types of conformance or other characteristics of semantic technologies (e.g., robustness and scalability).

Up to now, semantic technology conformance evaluations have been performed in terms of tool validation, being the two main efforts to this end those of the W3C ontology language specifications and of the RDF(S) and OWL Interoperability Benchmarking activities.

The *W3C ontology language specifications* include definitions of test cases for RDF(S) (Grant and Beckett, 2004), OWL (Carroll and Roo, 2004) and OWL 2 (Smith et al., 2009), which illustrate the correct usage of the ontology languages and the resolution of issues considered by the Working Groups. These test cases mainly cover conformance with regards to the ontology language semantics but also cover ontology language model and serialization conformance, both with correct and incorrect ontologies. Besides, the test cases are described in terms of ontologies to support the automation of their execution; however, software support is only provided to execute the OWL 2 test cases.

The *RDF(S) and OWL Interoperability Benchmarking activities* (García-Castro and Gómez-Pérez, 2009, 2010) involved the evaluation of the interoperability of semantic technologies using an interchange language and included a conformance evaluation with the goal of evaluating the conformance of semantic technologies with regards to an ontology language model.

During a conformance evaluation, described in detail in García-Castro and Gómez-Pérez (2010), a common group of tests is executed and each test describes one input ontology that has to be imported by the tool and then exported.

Each test execution comprises two steps, shown in Fig. 1. Starting with a file containing an ontology (O_i), the execution consists in importing the file with the ontology into the origin tool and then exporting the ontology to another file (O_i^{II}).

In these steps there is not a common way of checking how good the importers (by comparing O_i with O_i^I) and exporters (by comparing O_i^I with O_i^{II}) are. We just have the results of combining the import and export operation (the file exported by the tools), so these two operations are viewed as an atomic operation. It must be noted, therefore, that if a problem arises in one of these steps, we cannot know whether it was originated when the ontology was being imported or exported because we do not know the state of the ontology inside each tool.

After a test execution, we have two ontologies in the ontology representation language, namely, the original ontology (O_i) and the final ontology exported by the tool (O_i^{II}). By comparing these ontologies we can know up to what extent the tool conforms to the ontology language using the following metrics:

- *Execution* (*OK*/*FAIL*/*P.E.*) informs of the correct test execution. Its value is *OK* if the test is carried out with no execution problem; *FAIL* if the test is carried out with some execution problem; and *P.E.* (Platform Error) if the evaluation infrastructure launches an exception when executing the test.
- *Information added or lost* shows the information added to or lost from the ontology. We can know this information by comparing the original ontology with the final one; this comparison is performed both at the structural level and at the semantic level.
- *Conformance* (*SAME*/*DIFFERENT*/*NO*) explains whether the ontology has been processed correctly with no addition or loss of information. From the previous basic metrics, we can define *Conformance* as a derived metric that is *SAME* if *Execution* is *OK* and *Information added* and *Information lost* are void; *DIFFERENT* if *Execution* is *OK* but *Information added* or *Information lost* are not void; and *NO* if *Execution* is *FAIL* or *P.E.*

Two test suites were used to evaluate conformance, including only correct ontologies and covering the RDF(S) and OWL Lite languages. As in the case of the W3C test cases, the test suites were described using ontologies and the IBSE[2] tool was provided to automatically evaluate tools.

[2] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/ibse/.

The tests suites were defined manually and only contained 82 tests each. This, on the one hand, does not allow making an exhaustive evaluation of the conformance of the tools in terms of the ontology language model and, on the other hand, is costly and prone to errors.

To increase the exhaustiveness of conformance evaluations, we need a scalable way of generating conformance test data. Currently, some synthetic ontology generators are available (e.g., the Lehigh University Benchmark one (Guo et al., 2005)); however, using them for conformance evaluation data is not a good alternative because they are not exhaustive, since they cover a predefined part of the specification regardless of the number or size of ontologies generated, and they are not customizable in terms of the coverage to the ontology language specification.

3. Overview of the test suite generation process

In this paper we follow a keyword-driven approach for generating test suites for semantic technology conformance evaluations that is inspired in keyword-driven testing, a technique from the Software Testing area (Fewster and Graham, 1999). In this testing technique, tests are specified by describing the sequence of tasks to be performed in them using keywords. Each keyword, besides encapsulating a sequence of tasks, receives a set of parameters as input to allow reusing the keyword in different tests.

Inspired by the abovementioned technique, we have defined a keyword-driven test suite generation process that can be followed to produce conformance test suites. An overview of this process can be seen in Fig. 2.

A Keyword Library is used during the whole process; this library contains the definition of all the keywords that can be used to define conformance tests. In our case, these keywords define combinations of ontology components that can be composed to build the ontology that will be used in the test.

The process starts with a test suite definition script that contains all the tests defined in terms of keywords from the Keyword Library. The test suite generator takes this script and, first, the Preprocessor makes the script suitable to be interpreted and, second, for each test in the test suite the Interpreter reads each keyword in turn and constructs the defined ontology.

The output of the process is a test suite defined from the input script that contains, on the one hand, the ontologies defined for each test and, on the other hand, the metadata that describe the test suite and its tests. These metadata support the automated management of the test suite.

4. Definition of the Keyword Library

The keywords included in the Keyword Library have been extracted from the OWL abstract syntax grammar (Patel-Schneider et al., 2004), which defines the production rules that can be used to generate OWL DL ontologies.

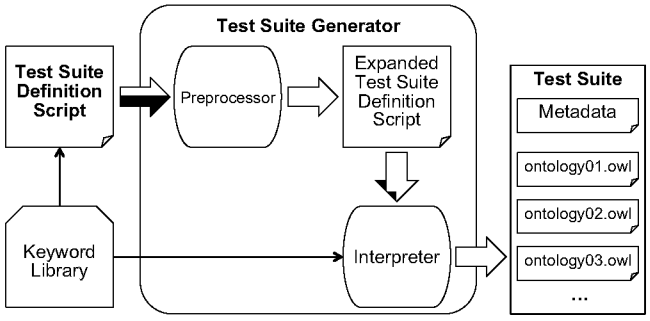

Fig. 2. Overview of the test suite generation process.

For every production rule in the OWL abstract syntax, we have defined different keywords taking into account the following rules:

- Alternatives in production rules (... | ...) have been covered by defining different keywords for each alternative symbol.
- In the case of optional symbols ([...]), separate keywords have been defined to include the case where the symbol is present and that where it is not.
- Symbol repetitions ({...}) have only been defined once in keywords; these repetitions can be inserted in scripts by repeating a keyword multiple times.
- When a production rule only includes a non-terminal symbol, the keyword combines the production rule and the rules of the non-terminal symbol.

Finally, keywords with a similar meaning have been grouped into a single keyword. For example, instead of defining a keyword for adding a domain to an object property and another for adding a domain to a datatype property, we have defined a generic keyword to add a domain to any type of property.

While grouping keywords decreases the number of keywords and simplifies the definition of test scripts, it also allows defining incorrect test scripts. This is not the only way of including modeling errors in scripts (e.g., using a string instead of a non-negative integer in a cardinality restriction). However, we leave to the script Interpreter the task of detecting these problems and notifying them.

Table 1 enumerates the keywords defined from the OWL abstract syntax with their corresponding parameters. They are classified into:

- *Restriction keywords*, which create classes from value and cardinality restrictions over properties.
- *Class description keywords*, which create named classes, enumerated classes and classes defined using set operators.
- *Property description keywords*, which create object, datatype and annotation properties as well as define property domains and ranges.
- *Property characteristics keywords*, which define global cardinality restrictions and logical characteristics in properties.
- *Individual description keywords*, which create named and anonymous individuals.
- *Axiom keywords*, which relate classes, properties and individuals using properties.
- *Data range keyword*, which creates an enumerated data range.
- *Annotation property keywords*, which define annotations over ontologies and ontology resources.
- *Ontology keyword*, which defines a descriptor for the ontology so it can be used in the script.

In order to build complex scripts using keywords, we need to reuse the results of a keyword in other keywords. Because of this, some keywords have a parameter (*resultId*) to identify inside the script the ontology components defined by a keyword. Besides, the value of the *origClassId* parameter in the first two keyword groups can be anonymous (if it starts with "_ANON") to create the class with an anonymous class description.

A simple example of a keyword definition can be found in the *createNamedClass* keyword, which can be used to create a named class and receives two parameters: the identifier of the keyword results that can be used inside the script (*resultId*) and the class name (*className*). Providing a full description of every keyword

Table 1
Keywords defined in the Keyword Library and their parameters.

Keyword	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Restriction keywords				
createClassAllValuesFromRestriction	resultId	origClassId	propId	classId
createClassSomeValuesFromRestriction	resultId	origClassId	propId	classId
createClassHasValueRestriction	resultId	origClassId	propId	value
createClassCardinalityRestriction	resultId	origClassId	propId	cardinality
createClassMinCardinalityRestriction	resultId	origClassId	propId	cardinality
createClassMaxCardinalityRestriction	resultId	origClassId	propId	cardinality
Class description keywords				
createNamedClass	resultId	className		
createClassUnion	resultId	origClassId	classId1	classId2
createClassIntersection	resultId	origClassId	classId1	classId2
createClassComplement	resultId	origClassId	classId	
createClassEnumerated	resultId	origClassId	indivId1	indivId2
Property description keywords				
createObjectProperty	resultId	propName		
createDatatypeProperty	resultId	propName		
createAnnotationProperty	resultId	propName		
addPropertyDomain	propId	classId		
addPropertyRange	propId	classId		
Property characteristic keywords				
makeObjectPropertyTransitive	propId			
makeObjectPropertySymmetric	propId			
makeObjectPropertyFunctional	propId			
makeObjectPropertyInverseFunctional	propId			
makeDatatypePropertyFunctional	propId			
Individual description keywords				
createIndividual	resultId	classId	indivName	
createAnonymousIndividual	resultId	classId		
Axiom keywords				
addObjectPropertyToClasses	origClassId	destClassId	propId	
addObjectPropertyToProperties	origPropId	destPropId	propId	
addObjectPropertyToIndividuals	origIndivId	destIndivId	propId	
addDatatypePropertyToIndividual	indivId	literal	propId	
Data range keywords				
createOneOfDataRange	resultId	literal1	literal2	
Annotation property keywords				
addAnnotationURI	resourceId	annPropId	URI	
addAnnotationLiteral	resourceId	annPropId	literal	
addAnnotationIndividual	resourceId	annPropId	indivId	
addOntologyAnnotation	ontPropId	ontURI		
Ontology keywords				
defineOntologyDescriptor	ontologyId			

and its corresponding parameters is out of the scope of this paper. Nevertheless, all the keywords are fully documented online.[3]

5. The test suite definition script

A *test suite definition script* is a comma-separated values (CSV) file that contains test definitions and macro definitions. Macros have been included to simplify the writing of test suites by defining test fragments that can be reused in different tests.

The CSV format was chosen to represent test suite definition scripts since it can be generated and edited using any text editor or spreadsheet software; these tools are well known by end users of semantic technologies who are the target users of the test suite generator.

Fig. 3 presents an example of a test suite definition script that includes one macro and two tests. As shown in the figure, the "*#MACROS#*" and "*#TESTS#*" lines mark the beginning of the macro and test definitions. Besides, lines starting with the hash symbol ('#') are considered as comments.

```
#MACROS#

Enumerated;resultId;originClassName;className;indivName1;indivName2
createNamedClass;_CD01;originClassName
createNamedClass;_CD02;className
createIndividual;_ID01;_CD02;indivName1
createIndividual;_ID02;_CD02;indivName2
createClassEnumerated;resultId;_CD01;_ID01;_ID02

#TESTS#
#This is a comment

*
createNamedClass;_result01;class01

*
Enumerated;_result01;class01;class02;individual01;individual02
```

Fig. 3. Sample test suite definition script.

A macro definition starts with a line containing the macro header which, first, includes the keyword that names the macro and, second, the different macro parameters. The rest of the macro definition is a

[3] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/OWLDLGenerator/OWLDLKeywords/.

sequence of one or more lines, each containing a keyword statement with the variables used to instantiate it. Fig. 3 shows the *Enumerated* macro, which creates a class defined by the enumeration of two individuals.

A test definition starts with a line containing either the identifier of the test or the asterisk ('*') character to generate these identifiers automatically during the interpretation of the test. The rest of the test definition is a sequence of one or more lines, each containing a keyword or a macro statement with the variables used to instantiate it. Fig. 3 includes two test definitions, one that creates an ontology containing a named class using the *createNamedClass* keyword and another that creates an ontology that contains an enumerated class using the macro previously presented.

To define a test suite, a user just has to define the different tests that compose it using the predefined keywords or the user-defined macros and by assigning values to their parameters. These values can be defined by the user or they can be the names of the RDF(S) and OWL built-in classes and properties qualified with their namespace abbreviation (e.g., "*owl:equivalentClass*").

It must also be noted that, even if the definition of tests is expected to be performed manually, in some cases it may be convenient to automate this definition using some simple program or script (e.g., to generate tests for all the XML Schema Datatypes or tests that cover all the built-in annotation properties).

6. The test suite generator

As shown in Fig. 2, the test suite generator has two main modules, namely, a *Preprocessor* module and an *Interpreter* module.

Once the *Preprocessor* module receives a test suite definition script, it has to preprocess the script so it can be correctly managed by the Interpreter. This preprocessing consists in replacing each macro statement appearing in a test definition with the macro definition and instantiating the corresponding macro parameters with the values from the macro statement.

The *Interpreter* module interprets the preprocessed test suite definition script in order to generate, on the one hand, the metadata that describes the test suite and each of the tests included in it and, on the other hand, the data to be used in each test, which is an OWL ontology file. This OWL ontology file will serve both as input for the test execution and as expected result for the analysis of the test result.

The Interpreter contains implementations for every keyword in the Keyword Library (see Section 4); these implementations create the combinations of ontology components defined by the keyword using an ontology management library.

For each test in a test suite definition script, the Interpreter sequentially executes the implementation of each keyword statement in the test definition in order to create the ontology defined by the test. Simultaneously, it collects the information needed to generate the test metadata.

As mentioned above, the test suite generator describes its outputs (a test suite and the tests that compose it) according to certain metadata schemas. Currently, these schemas are implemented using two OWL ontologies: the *TestSuite* ontology, which defines the vocabulary to represent tests and test suites in general, and the *ImportTestSuite* ontology, which specializes the previous ontology to represent tests and test suites that evaluate semantic technology conformance.

These ontologies are available online,[4] are lightweight (since their main goal is to be user-friendly) and allow providing a machine-processable description of the test suite to support the automation of evaluation tasks.

The test suite generator has been developed using Java, and Jena[5] is the ontology management library used to implement keywords. It is also available in the Web.[6] Once downloaded, it can be used by passing the name of the test suite definition script as a parameter and will produce the ontology and metadata files that compose the test suite defined in the script.

7. The OWL DL import test suite

This section presents how the OWL DL Import Test Suite was defined by, first, creating a test suite definition script using the keywords presented in Section 4 and, second, running the test suite generator described in Section 6.

Our design principles when defining this test suite were: to define only simple ontologies, so problems can be easily identified in tools; to define only correct ontologies, because our goal is not to analyze the behavior of the tools with wrong or inconsistent ontologies; and to use the RDF/XML syntax for serializing ontologies, since it is the one used and recommended to interchange ontologies.

Besides, while the exhaustiveness of a test suite for evaluating conformance is highly desirable, its efficiency is an aspect not to disregard. Therefore, we avoided an indiscriminate definition of tests and followed some guidelines to try to maximize as much as possible these opposing properties:

(a) To use all the keywords defined in the Keyword Library.
(b) For all the keywords that have a class description as parameter, to test all the possible ways of describing classes. If the keyword has more than one class description parameter, tests have been defined only for one parameter.
(c) To cover all the different combinations of resources using a property.
(d) To create classes defined with set operators and restrictions as anonymous classes instead of as named classes.

By applying these guidelines we defined 561 tests to cover all the simple combinations of components of the OWL DL knowledge model and classified them into the following groups:

- *Class descriptions* (74 tests), which contains tests for class descriptions defined using: named classes, enumerated classes, value and cardinality constraints in object and datatype properties, and set operators. The group, besides elementary class descriptions, includes tests for: value constraints with restrictions in terms of a class description (named class or class description) and of a built-in data range; the *owl:hasValue* constraint with individual and literal values; cardinality constraints with cardinality values of 1 and 2; and set operators with named classes and class descriptions.
- *Class axioms* (96 tests), which contains tests for class axioms defined by two classes related by one of the built-in properties for class axioms that have a class description either as subject or as object.
- *Combinations of class axioms* (14 tests), which contains tests for the different combinations of classes using the built-in properties for class axioms except that of a class being disjoint with itself.

[4] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/OWLDLGenerator/ontologies/.

[5] http://jena.sourceforge.net/.

[6] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/OWLDLGenerator/.

- *Property descriptions* (eight tests), which contains tests for object, datatype, and annotation properties, logical characteristics of properties, and global cardinality restrictions on properties.
- *Combinations of property axioms* (24 tests), which contains tests for the different combinations of properties using the built-in properties for property axioms except that of an object property being inverse of itself and that of a datatype property being inverse of any property.
- *Properties with domain and range* (58 tests), which contains tests for object properties that have a class description as domain or as range, for datatype properties that have a class description as domain, and for the different combinations of object and datatype properties with domains and ranges.
- *Individual descriptions* (38 tests), which contains tests for named and anonymous individuals that are instance of a class description.
- *Combinations of individual axioms* (nine tests), which contains tests for the different combinations of individuals using the built-in properties for individual axioms except that of an individual being different from itself.
- *Individuals and properties* (11 tests), which contains tests for the different combinations of named and anonymous individuals that are related to other individuals using object properties and that are related to values using datatype properties.
- *Data ranges* (76 tests), which contains tests for enumerated datatypes and for the XML Schema Datatypes that can be used in OWL (Patel-Schneider et al., 2004). Tests for enumerated datatypes include when they are the range of a property (both with and without domain) and when a named or an anonymous individual is related to a value of an enumerated datatype. Tests for XML Schema Datatypes include when they are the range of a property (both with and without domain).
- *Annotation properties* (153 tests), which contains tests for ontology annotation properties and for annotation properties with individual, literal and URL values for the different resources: ontologies, classes, datatypes, object and datatype properties, annotation properties, and individuals. Tests are defined for the built-in ontology annotation properties, and for user-defined annotation properties.

Not all the 561 tests were defined manually; we defined 40 macros to simplify test definition and implemented scripts to generate those tests where only one of the parameters changed (e.g., to cover all the XML Schema Datatypes). This can be seen in the test suite definition script used to generate the test suite, which can be found online.[7]

Compared to the existing related test suites (Section 2), the OWL DL Import Test Suite is an extension of the OWL Lite test suite used in the OWL Interoperability Benchmarking; therefore, it completely covers the OWL Lite test suite and adds new tests for OWL DL.

When comparing the OWL DL Import Test Suite with the W3C OWL test cases, the main difference is that the former mainly deals with conformance in terms of the ontology language model while the latter mainly comprises tests covering conformance in terms of the ontology language semantics.

Consequently, while the test suite presented in this section contains simple and correct ontologies that use few OWL components, the OWL test cases primarily deal with entailment and consistency checking using complex examples (e.g., for comparing the RDF(S) and OWL semantics).

[7] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/OWLDLGenerator/OWLDLTSScript.csv.

Besides, the OWL DL Import Test Suite only covers 27 of the 169 ontologies used in the OWL test cases[8]; therefore, both test suites can be seen as complementary.

8. Using the OWL DL Import Test Suite

Once the OWL DL Import Test Suite was generated, we proceeded to evaluate some tools with it. We selected the Protégé[9] ontology engineering tool in its two last incarnations, Protégé 3.5 alpha (build 644) with the OWL plugin and Protégé 4.2 beta (build 269), since it is the ontology engineering tool with the largest user base and from one version to the other there has been a significant change in the way of managing OWL ontologies (the ontology management library changed from the Protégé-OWL API to the OWL API).

We wanted to analyse whether we were able of identifying changes in the management of OWL DL ontologies between the two tool versions and whether these changes could have any effect from the user point of view. To obtain this information, we followed the conformance evaluation defined for the OWL Interoperability Benchmarking (presented in Section 2) and automated the evaluation using the IBSE tool.

Table 2 presents a summary of the results of the two versions of Protégé when evaluating their conformance using the OWL DL Import Test Suite. We can see that the ontologies produced by the tools are semantically equivalent to the ones in the test suite with some exceptions.

Regarding Protégé 3.5 alpha, the cases when the ontologies are different are those when the ontology contains:

- A literal value. The literal value is created with a datatype of *xsd:string* and, therefore, it is a different literal. According to the RDF specification, one requirement for literals to be equal is that either both or neither have datatype URIs.[10]
- Class descriptions that are the subject or the object of a *rdfs:subClassOf* (or of an *owl:disjointWith*) property. In these cases, the class description is defined to be equivalent to a new class named "Axiom0", and this new class is the subject or the object of the *rdfs:subClassOf* (or the *owl:disjointWith*) property.

Thanks to the test suite generator, it is easy to go deeper into the results. For example, this new class created by Protégé 3.5 alpha is named "Axiom0" and we want to know whether there would be any problem if the previous situation happened twice in the ontology or if a class with that name already existed. Within seconds we write a couple of tests by extending a previous one to find this out, shown in Fig. 4, and we check that the tool detects that a class with that name already exists and changes the name of the new class to "Axiom1".

Regarding Protégé 4.2 beta, its results highly depend on the ontology management library that it uses (the OWL API version 3.2.5 1928). When the OWL API processes OWL DL ontologies, it converts the ontologies into OWL 2. Since OWL 2 covers the OWL DL specification, most of the times the OWL API produces equivalent ontologies. However, one effect of this conversion is that individuals are converted into OWL 2 named individuals.

The cases when the ontologies are different are those when the ontology contains:

[8] This calculation disregards those tests that use incorrect ontologies and OWL Full, and just takes into account premise ontologies in those tests with premises and conclusions (i.e., entailment tests).

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

[10] http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#dfn-typed-literal.

Table 2
Conformance results for Protégé 3 and Protégé 4.

Conformance	Protégé 3.5 alpha	Protégé 4.2 beta[a]
Semantically equivalent	429	559
Semantically different	132	2
Execution fails	0	0
Total	561	561

[a] Not counting additions of *owl:NamedIndividual*.

```
*
createNamedClass;_CD01;class01
createNamedClass;_CD02;class02
createClassComplement;_CD03;_ANON01;_CD02
createLinearClassCombination rdfs:subClassOf;_CD03;_CD01
createNamedClass;_CD04;class03
createNamedClass;_CD05;class04
createClassComplement;_CD06;_ANON02;_CD05
createLinearClassCombination;rdfs:subClassOf;_CD06;_CD04

*
createNamedClass;_CD00;Axiom0
createNamedClass;_CD01;class01
createNamedClass;_CD02;class02
createClassComplement;_CD03;_ANON01;_CD02
createLinearClassCombination;rdfs:subClassOf;_CD03;_CD01
```

Fig. 4. New tests defined for Protégé 3.5 alpha.

- Datatype properties with range most of the time-related XML Schema datatypes (i.e., *date*, *gDay*, *gMonth*, *gYear*, *gYearMonth*, *time*). In these cases the range is defined as an *rdfs:Datatype*.[11]
- Datatype properties with range *xsd:gMonthDay*. In these cases datatype properties are created both as datatype properties and as object properties; besides, the range is defined as an *owl:Class*.

In previous evaluations (García-Castro and Gómez-Pérez, 2010) Protégé 3 with the OWL plugin posed no conformance problems. Now we have detected some cases where the tool changes the conceptualization of the ontology but that do not affect the overall semantic model of the ontology or any instances created using the ontology. On the other hand, Protégé 4 poses some problems when managing certain component combinations; however, we must note that the version that we have evaluated is in "beta" state, and we expect to find no problems in the final release.

Now, let us suppose that in a certain use case we frequently model classes with labels and we want to cover this more exhaustively in the evaluation. Again, it is straightforward to define a new *createNamedClassWithLabel* macro, shown in Fig. 5, and to replace in the script all the occurrences of the *createNamedClass* keyword by this new macro. When evaluating the two tools with this new test suite[12] we obtain identical results (see Table 2) and conclusions as before.

9. Discussion

With the approach presented in this paper, we aim to increase the exhaustiveness of conformance evaluations by providing an scalable way of defining test data instead of by randomly generating these data. Although the need for random evaluation data generation is acknowledged in some situations (Hamlet, 2006), the general belief is that the random generation of input data is the least effective approach (Myers et al., 2004).

```
createNamedClassWithLabel;resultId;className
createNamedClass;resultId;className
addAnnotationLiteral;resultId;rdfs:label;"className"@en
```

Fig. 5. New *createNamedClassWithLabel* macro.

It could be argued that users could generate their conformance evaluation ontologies using the OWL abstract or functional syntaxes or by directly using an ontology management library. Defining the ontologies in terms of the presented keywords is much more simpler than using one of these alternatives. Furthermore, in the case of ontology management libraries, they include too many classes and methods and require development expertise.

Besides, the use of the Keyword Library and the Test Data Generator also provides as a side effect some minimal syntax-checking capabilities (according to the OWL specification) when defining evaluation data. Since one of the drawbacks of the manual definition of evaluation data is the apparition of unintended errors in such data, these syntax-checking capabilities help reducing these errors.

Another point that we cannot disregard is the influence of the tools used during the generation of evaluation data in the evaluation results. Clearly, the ontology management library used in the test suite generator and other tools that use this library for ontology parsing should pose less or no problems when managing the generated ontologies.

In order to objectively evaluate these tools, the Interpreter module should have an alternative implementation that used another ontology management library. The keyword-driven approach that we follow makes a distinction between the definition of the test suite and the implementation of the generator, which is useful in this case where we want to maintain the test suite definition scripts independent from the ontology management library used in the Interpreter.

10. Conclusions and future work

This paper shows how we have applied a keyword-driven approach to the generation of test suites for evaluating ontology language conformance.

Our test suite generation process, supported by the implemented test suite generator, facilitates defining conformance test suites by users and significantly reduces the effort of doing it. This time, we managed to define the 561 tests of the OWL DL Import Test Suite in terms of days instead of in terms of weeks, what happened in the definition of the previous test suites for RDF(S) and OWL Lite (which contained only 82 tests each).

Nevertheless, the definition of a test suite such as the one presented in this paper still requires an understanding of the OWL language as well as a significant effort for defining correct and efficient tests. As an example of this, in Section 8 we have seen how doubling the number of tests (by including classes with labels) produced no further significant conclusions and defining two focused tests provided useful insights of the tool.

Regardless of test efficiency, which depends on the person defining tests, the point to remark is that the approach presented in this paper allows scaling the definition of evaluation data, on the one hand, by permitting the parameterized generation of test suite definition scripts and, on the other hand, by allowing the reuse of existing test suite definitions to build incremental test suites.

[11] These cases are counted in the "semantically equivalent" row in the table.
[12] The tests that do not involve named classes are identical to those in the original test suite.

Up to now, we have been the only users of the test suite generator. By freely distributing it, we plan to obtain feedback from real users about the usability and efficiency of our approach in order to improve it.

In addition, the test suite generator is open to some improvements that have already been mentioned, such as providing syntax-checking capabilities during test definition or an alternative implementation of the Interpreter module with other ontology management library.

Even if the test suite generator has been defined in the scope of conformance evaluations, the test suites that it produces can be used in other types of evaluations (e.g., interoperability). Besides, the approach here presented can be reused to generate test suites for other ontology languages, such as OWL 2. In this case, the method presented in Section 4 could be applied to extract keyword definitions from the OWL 2 functional-style syntax (Motik et al., 2009).

To achieve a systematic evaluation of the conformance of semantic technologies we need to automate as much as possible the whole conformance evaluation process. Previous work covered the task of executing the evaluation and with this work we support the definition of new evaluation data. The next step will be to automate the analysis of conformance results; to this end, the test suite generator already includes information in the test suite metadata to facilitate the categorization of tests (and their results) according to the combinations of components covered by their ontologies. Furthermore, these metadata enable using the ontologies produced by the test suite generator in existing evaluation frameworks (Gardiner et al., 2006; Babik and Hluchy, 2008) and infrastructures (García-Castro et al., 2010).

References

Babik, M., Hluchy, L., 2008. A testing framework for OWL-DL reasoning. In: Proceedings of the 4th International Conference on Semantics, Knowledge and Grid (SKG 2008). IEEE Computer Society, Beijing, China, pp. 42–48.

Burnstein, I., 2003. Practical Software Testing: A Process-oriented Approach. Springer Verlag.

Carroll, J., Roo, J.D., 2004. OWL Web Ontology Language Test Cases. Tech. Rep. W3C Recommendation, 10 February 2004 ⟨http://www.w3.org/TR/owl-test/⟩.

Fewster, M., Graham, D., 1999. Software Test Automation: Effective Use of Test Execution Tools. ACM Press/Addison Wesley Publishing Co., New York, USA.

García-Castro, R., Gómez-Pérez, A., 2009. RDF(S) interoperability results for Semantic Web technologies. Int. J. Software Eng. Knowl. Eng. 19 (8), 1083–1108.

García-Castro, R., Gómez-Pérez, A., 2010. Interoperability results for Semantic Web technologies using OWL as the interchange language. Web Semantics Sci. Serv. Agents World Wide Web 8 (4), 278–291.

García-Castro, R., Esteban-Gutiérrez, M., Gómez-Pérez, A., 2010. Towards an infrastructure for the evaluation of semantic technologies. In: Proceedings of the eChallenges e-2010 Conference (e-2010), Warsaw, Poland, pp. 1–8.

Gardiner, T., Tsarkov, D., Horrocks, I., 2006. Framework for an automated comparison of description logic reasoners. In: Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Lecture Notes in Computer Science, vol. 4273. Springer, Athens, GA, USA, pp. 654–667.

Grant, J., Beckett, D., 2004. RDF Test Cases. Tech. Rep. W3C Recommendation, 10 February 2004 ⟨http://www.w3.org/TR/rdf-testcases/⟩.

Guo, Y., Pan, Z., Heflin, J., 2005. LUBM: a benchmark for OWL knowledge base systems. Web Semantics Sci. Serv. Agents World Wide Web 3 (2), 158–182.

Hamlet, D., 2006. When only random testing will do. In: Proceedings of the 1st International Workshop on Random Testing (RT 2006), Portland, USA, pp. 1–9.

Motik, B., Patel-Schneider, P.F., Parsia, B., 2009. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax, Tech. Rep. W3C Recommendation, 27 October 2009 ⟨http://www.w3.org/TR/owl2-syntax/⟩.

Myers, G., Sandler, C., Badgett, T., Thomas, T., 2004. The Art of Software Testing, second edition Wiley.

Patel-Schneider, P.F., Hayes, P., Horrocks, I., 2004. OWL Web Ontology Language Semantics and Abstract Syntax. Tech. Rep. W3C Recommendation, 10 February 2004 ⟨http://www.w3.org/TR/owl-semantics/⟩.

Smith, M., Horrocks, I., Krötzsch, M., Glimm, B., 2009. OWL 2 Web Ontology Language Conformance. Tech. Rep. W3C Recommendation, 27 October 2009 ⟨http://www.w3.org/TR/owl2-test/⟩.