



Semantic Repository for RDF(S) and OWL

Tests and Benchmarks

12 June 2007

SwiftOWLIM v.2.9.0 and BigOWLIM v.0.9.4

1	Introduction.....	2
2	Configuring OWLIM and Sesame.....	2
3	Running Entailment Test Cases.....	3
3.1	JUnit Entailment Tests	3
3.2	W3C Entailment Tests.....	7
4	Running EXQUANT and UNION tests.....	8
5	Running the Wordnet Sample Application	9
6	Running Lehigh University Benchmark (LUBM).....	10
7	Running OUBM	10
8	References	12

1 Introduction

OWLIM is a semantic repository, packaged as a Storage and Inference Layer (SAIL) for Sesame RDF database. More information on OWLIM can be found at <http://www.ontotext.com/owlim>. More information about the inference and persistency strategies, configuration, installation and performance of OWLIM can be found in the system documentation respectively of SwiftOWLIM, [10], and BigOWLIM, [11].

Both versions of OWLIM are packaged as storage and inference layers (SAIL) of the Sesame RDF database, <http://www.openrdf.org>. Thus, all the test applications are designed to use the corresponding Sesame APIs. Switching from one version of OWLIM to another, or to a completely different Sesame backend, is a matter of change of Sesame's configuration. Thus, OWLIM's test applications can be used for evaluation and validation of the capabilities of a number of other backend options.

The two versions of OWLIM bear identical reasoning capabilities and have many of their configuration parameters in common. An overview of the differences in their performance and behavior can be found in each of the system documentations, [10] and [11].

This document describes the various tests included in OWLIM's distributions and gives information on how to setup and run them. For the user's convenience, the distribution includes several batch files that can be used to run the different tests. Running the test and the benchmarks may require minor modifications in the scripts and configuration files, as discussed in section 2.

Requirements for all included tests:

- Java v1.4 or higher;
- Sesame v. 1.2.1 to 1.2.6.

2 Configuring OWLIM and Sesame

Running the tests (as well as the sample applications of OWLIM), requires modification of the script `setvars` (respectively `.cmd` and `.sh`) in OWLIM's main folder. The most important setting is the specification of the Java virtual machine, though the `JAVA_HOME` environment variable.

Running properly some of the test cases, may require modification of the specifications of the alternative repository configurations in the configuration file of Sesame. All test applications share a single configuration file, named `system.conf`, which can be found (and must stay located) in the `test/bin/test` sub-directory of OWLIM's main directory. The configuration file's content is divided in two parts: system configuration (contains user names and passwords); and repository list, which contains information about the configuration of each repository. This mechanism allows different OWLIM configurations to be used or tested for the different tests and benchmarks.

The config file contains a repository `test`, defined as follows:

```
<repository id="test">
  <title>Test Entailment repository </title>
  <sailstack>
    <sail class="org.openrdf.sesame.sailimpl.OWLIMSchemaRepository">
      <param name="file" value="./kb/test.nt"/>
    </sail>
  </sailstack>
</repository>
```

```

<param name="dataFormat" value="ntriples"/>
<param name="compressFile" value="no"/>
<param name="ruleset" value="owl-max"/>
<param name="partialRDFS" value="true"/>
<param name="indexSize" value="4000000"/>
<param name="new-triples-file"
value="./kb/new-temp-triples.nt"/>
<param name="base-URL"
value="http://example.org/entailment/test#"/>

<!-- semicolon should be used as delimiter -->
<param name="imports" value=
"/kb/owl/owl.rdfs;
./kb/test/testSchema.rdf;
./kb/test/testOntology.rdf;"/>
<param name="defaultNS" value=
"http://www.w3.org/2002/07/owl#;
http://www.w3.org/2000/10/rdf-
tests/rdfcore/testSchema#;
http://www.w3.org/2002/03owlt/testOntology#;"/>

</sail>
</sailstack>

<!--ACLlist can contain zero or more 'user' elements-->
<acl worldReadable="true" worldWritable="true">
  <user login="admin" readAccess="true" writeAccess="true"/>
</acl>
</repository>
    
```

This repository uses `org.openrdf.sesame.sailimpl.OWLIMSchemaRepository` as a SAIL, which predetermines the usage of SwiftOWLIM. It stores the data in N-Triple format in the file `test/kb/test.nt`. The temporary statements are stored in the file `test/kb/new-temp-triples.nt`. Its base URL is `http://example.org/entailment/test#` and imports `owl.rdfs`, `testSchema.rdf` and `testOntology.rdf` with default namespaces `http://www.w3.org/2002/07/owl#`, `http://www.w3.org/2000/10/rdf-tests/rdfcore/testSchema#` and `http://www.w3.org/2002/03owlt/testOntology#` respectively.

3 Running Entailment Test Cases

There are two different kinds of entailment tests provided with OWLIM:

- JUnit tests for regression testing of OWLIM's reasoning support;
- W3C's Normative OWL entailment test, [3].

3.1 JUnit Entailment Tests

This is OWLIM's own regression test framework for inference. It uses a set of OWL files, each of which contains few statements to be asserted. After loading each of the files the test checks whether or not a given set of statements is present in the inferred closure, after some hard-coded logic. The test itself does not rely on any ontology, apart from the OWL schema which is loaded by default. The repository configuration used in these tests has id "owlim", specified in the `test/bin/test/system.conf` file.

To execute the tests, run the `junit-test.cmd(.sh)` script. Note that it is configured to use `junit.swingui.TestRunner` frontend, so in case you are running it in a textmode environment, change the script so to use `junit.textui.TestRunner` instead.

The source code of this JUnit test suite can be found in `src/test/sesame` distribution folder.

`TestOWLIMRepository` class uses the JUnit library. Each document is used in a separate test case:

- `owl_test1a` – a symmetric property test: if a property `symmetricTestProp` is an `owl:SymmetricProperty` and there is a statement `<John symmetricTestProp Mary>`, then the statement `<Mary symmetricTestProp John>` must be present in the repository. Similarly, if the statement `<symmetricTestProp rdf:type owl:SymmetricProperty>` is removed from the repository, then the statement `<Mary symmetricTestProp John>` must not be present;
- `owl_test2a` – a transitive property test: if a `transitiveTestProp` is an `owl:TransitiveProperty` and the repository contains the statements `<Mary transitiveTestProp Harry>` and `<John transitiveTestProp Mary>`, then it must also contain the statement `<John transitiveTestProp Harry>`. Similarly, if one of these statements is removed, then the last statement must be removed as well and if the removed statement is added anew, the last statement must again be added;
- `owl_test1a2a` – defines `transitiveAndSymmetricTestProp`, which is both `owl:SymmetricProperty` and `owl:TransitiveProperty`. `Mary` and `Harry`, as well as `John` and `Mary`, are connected by means of this relation. The test checks if the full symmetric and transitive closure is inferred.
- `owl_test3a` – defines `inverseOfTransitive` property which is `owl:inverseOf owl:TransitiveProperty`. `John`, `Mary` and `Harry` are `owl:transitiveTestProp` in this order as it was the case with `owl_test2a`. The test checks if they are `inverseOfTransitive` in this order: `Harry`, `Mary`, `John`. It also checks if any of them is `inverseOfTransitive` of itself (because the inferencer applies the transitivity `<a trans b>` and `<b trans c>` entails `<a trans c>` only when the three variables are distinct).
- `owl_test10a` – same as `owl_test2a`, except that it defines `eqTestProp` which is `owl:equivalentProperty` to `transitiveTestProp`. `owl:equivalentProperty` means that `eqTestProp` and `transitiveTestProp` might be interchangeably used. In the inferred closure all subjects and objects linked with `transitiveTestProp` must also be linked with `eqTestProp` and that is what the test checks.
- `owl_test10b` – test focused on `rdfs:domain` and `rdfs:range`. If `John`, `Mary` and `Harry` are related through `transitiveTestProp` in this order. `inverseOfTransitive` takes its subject from the class `DomainName`, while `eqTestProp` (`owl:equivalentProperty` to `inverseOfTransitive`) takes its object from the class `RangeName`. It follows that, `Harry` and `Mary` must belong to `DomainName`; `Mary` and `John` must belong to `RangeName`. Moreover, `Harry` must not be a member of `RangeName` and `John` must not be a member of `DomainName`, while `Mary` must belong to both classes.

- owl_test12a* – similar to *owl_test10b* with a slight difference, namely that `eqTestProp` is `owl:sameAs inverseOfTransitive` (. The difference comes from the fact that `owl:sameAs` stronger than `owl:equivalentProperty`; it will copy all the statements containing `inverseOfTransitive` and will replace it with `eqTestProp` no matter if the former appears somewhere as a subject, predicate or object. The results must be the same as that of *owl_test10b*.
- owl_test12b* – tests the `owl:sameAs` inference. If `John`, `Mary` and `Harry` are `transitiveTestProp` as in the example above and at the same time `<MaryAlias owl:sameAs Mary>`, then `MaryAlias` and `Mary` must be used interchangeably.
- owl_test15a*, *owl_test15b* – tests the `owl:FunctionalProperty` inference. According to rule `owl_FunctProp`, if a property `p` is `owl:FunctionalProperty` and at the same time `<x p y>` and `<x p z>`, it follows that `<y owl:sameAs z>` (i.e. the “value” of `p` for a given argument is unique, since `p` represents a function). To illustrate: if `John`’s mother is `Mary` and at the same time `John`’s mother is `Carmen`, then `Mary` and `Carmen` must be one and the same person as long as `hasMother` is an `owl:FunctionalProperty`. But if in a given moment `hasMother` ceases to be `owl:FunctionalProperty`, then the test should not entail that `Mary` and `Carmen` are one and the same person. The property `motherOf` is `owl:inverseFunctionalProperty`, which is also `owl:inverseOf hasMother`. According to rule `owl_InvFunctProp`, if `p` is `owl:inverseFunctionalProperty` and `<x p z>` and `<y p z>` are asserted, then `<x owl:sameAs z>` should be entailed. With the above conditions we have `<Mary motherOf John>` (the inverse of `<John hasMother Mary>`) and `<Carmen motherOf John>` (the inverse of `<John hasMother Carmen>`). And again, if in a given moment `motherOf` ceases to be `owl:inverseFunctionalProperty`, then `Mary` and `Carmen` must be different persons; in this case `hasMother` will cease to be `owl:FunctionalProperty`.
- owl_test16a*, *owl_test16b* – tests of `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`. The property `ownSSNumber` (Social Security Number) is defined as `owl:InverseFunctionalProperty`. If two persons have the same Social Security Number, it follows that they must be `owl:sameAs`. In *owl_test16b*, `ownSSNumber` is defined as `owl:inverseOf ownedBy`, which in its turn is `owl:FunctionalProperty`. And again, if two people have the same Social Security Number, it follows that they must be one and the same person (the two definitions of `ownSSNumber` are equivalent).
- owl_testgos1a* – tests a number of transitive properties – `partOf`, `subRegionOf` and `livesIn`. `Europe` and `Asia` are `partOf` the `Earth`; `Germany` and `Japan` are respectively `subRegion-s` of `Europe` and of `Asia`; `Berlin` is `subRegionOf Germany`. `partOf` is `owl:TransitiveProperty`, `subRegionOf` is `rdfs:subPropertyOf partOf` (but it is NOT `owl:TransitiveProperty partOf`), and `livesIn` is `protons:transitiveOver subRegionOf`. Then, if `<John livesIn Berlin>`, it follows that he `livesIn Europe`, but it is not true that he `livesIn Earth` (because `livesIn` is `protons:transitiveOver subRegionOf`, but `subRegionOf` itself is not defined as `owl:TransitiveProperty`, although it is `rdfs:subPropertyOf partOf`).

- *owl_test_oneOf* – tests whether an instance, specified within the declaration of an enumerated class `tst:TShirt` is made a member of this class, `tst:TShirt` – checked for the `tst:small` instance.
- *owl_minCardinality1* – the test class `tst:PersonWithEmail` is declared to be equivalent to a restriction over property `tst:email` with `owl:minCardinality` with value of `"1"^^xsd:nonNegativeInteger`, then some instance, `tst:Pesho` that is related to a literal with the property `tst:email` is checked to be a member of `tst:PersonWithEmail` class.
- *Owl_TBOX_SomeValFrom* – this test checks for class subsumption based on `owl:someValuesFrom` restrictions over sub-properties to values of classes that are sub-classes. The test case has two named class definitions `tst:Employee` and `tst:ResearchAssistant` equivalent to such restrictions over `tst:worksFor` property and relating values of `tst:Organisation` and its subclass `tst:ResearchGroup`. The test checks whether an instance `tst:Naso`, which is related with the property `tst:workFor` to some instance of class `tst:ResearchGroup`, is a member of the `tst:Employee`.
- *Owl_TBOX_AllValFrom* – a test similar to the above but covering relationships between restrictions of type `owl:allValuesFrom`. Consult the test case OWL document – `owl_TBO_AVF.owl` for more details of the derived subsumption.
- *Owl_TBOX_hasValue* – a test similar to the two above, but covering relationships between two `owl:hasValue` restrictions.
- *Owl_TBOX_HasValSomeValuesFrom* – is a test that examines the relations between `owl:someValuesFrom` and `owl:hasValues` restrictions to derive subsumption.
- *Owl_TBOX_SomeValues_minCardinality1* – similar as the above ones, but now in the scope of the test is the relation between restrictions of type `owl:someValuesFrom` and `owl:minCardinality` with value 1.
- *Owl_test17* – test to check whether the semantics of `owl:unionOf` are supported correctly. It defines a class `Person` to be the union of `Man`, `Woman` and `Kid` classes and then defines a single instance of each of those and checks whether they are members also of the `Person` class.
- *Owl_test18* – test to check whether the semantics of `owl:intersectionOf` are supported correctly. It defines three top classes `Person`, `Male` and `Female` and then defines `Man` to be an intersection of `Person` and `Male` and `Woman` – of `Person` and `Female` classes. Correctness is checked in both directions of the definitions. First it declares an instance of `Man` and checks whether it is a member of both `Male` and `Person`. In the opposite direction, it declares an instance of `Male` and `Person` and checks if it is also a member of `Man`. The same test is made for an instance that is a member of `Female` and `Person` simultaneously.
- *Owl_test19* – test to check whether the semantics of `owl:oneOf` are supported correctly. Defines a class `Groul` which has exactly three members and then checks whether those instances are members of the enumeration `Group`.

Note: *TBOX and cardinality tests will pass only if the repository is configured with ruleset owl-max.*

The test also include testcases about "READ_COMMITTED" transactional behaviour, separate retrieval of explicit/implicit statements and multithreaded inference.

3.2 W3C Entailment Tests

Class `EntailmentTest` is a framework for checking of compliance with the, so called, approved tests that W3C provides in [3]. Each test is a set of premises and conclusions given as OWL documents. Depending on the kind of the test employed (described in the corresponding Manifest OWL document), the check determines whether or not the conclusions are present in the repository.

The following types of tests are described in `testSchema.rdf` and `testOntology.rdf`:

- **Positive entailment tests** – check if the inference engine infers the expected results;
- **Negative entailment tests** – check if the inference engine does infer a/the statement(s) listed in a separate file created especially for the test;
- **True test** – a test that checks if the repository contains a number of statements that are either axioms or inferred from axioms (and thus must be present in every repository, including in the "empty" ones);
- **OWL for OWL test** – test that illustrates how OWL is used to describe/for the description of OWL Full.
- **Import level tests** – serve to indicate if there is an interaction between `owl:imports` and the sublanguage elements in the main document (that imports certain ontologies);
- **Not OWL feature test** – a kind of negative test; checks if the inference engine supports features of DAML+OIL, which OWL does not support anymore;
- **Import entailment test** – checks if the premise document, and its "imports" closure, entails the conclusion document;
- **Inconsistency test** – a kind of positive entailment test that indicates if there are inconsistencies in the inferred closure of an ontology (e.g. if two entities are `owl:sameAs` while at the same time they are `owl:differentFrom` one another; if an entity is of type `owl:Nothing` or it is `rdfs:subClassOf owl:Nothing` and at the same time it is `owl:differentFrom owl:Nothing`. Statements of this kind are inconsistent with the semantics of OWL);
- **Consistency test** – a kind of negative entailment test; checks if the OWL document does entail a falsehood.

For now, the OWLIM entailment test supports the Positive and Negative Entailment tests only. They could be used to determine what part of OWL OWLIM inference engine is able to handle.

The OWL files with the test cases are available at <http://www.w3.org/TR/owl-test/> (as part of the file `approved.zip`). On the same web-site one can find more information on the features that each test checks.

After the test cases have been retrieved and unpacked in a specified folder, modify the `entailment-test` script (`.cmd` or `.sh`, respectively for Windows or Linux) so that `APPROVED_DIR` points to the folder, where the test cases reside.

When run, the script automatically passes over the contents of the subfolders of **APPROVED_DIR** and parses the **ManifestXXX** fiels found there. Then, it proceeds with a particular test case and delivers the results.

If the name of an **APPROVED_DIR**'s subfolder is specified as an optional script argument, then only the testcases contained in that particular subfolder will be performed. For example:

```
C:\owlim\test>entailment-test.cmd AllDifferent
```

has the following output:

```
Processing folder AllDifferent
positive test:(#Full): AllDifferent/Manifest001#test passed.
```

4 Running EXQUANT and UNION tests

Exquant, [12], is a benchmark that tests mainly the capability of systems to reason against long transitive chains. A complete class definition of class *c* states that, if an individual is related through a transitive property *p* to an instance of class *c*, it is itself a member of *c*. The test generates a chain with the specified length (in the results reported here 1 000 or 10 000) of individuals linked through *p*, and then asserts that the last individual in the chain is member of *c*. At the end a query, retrieving all the instances of *c*, is evaluated – this query will get complete results only if the transitivity of the property *p* and the definition of class *c* are correctly interpreted. As basic strategy of OWLIM is forward-chaining and total materialization, it do not perform well compared to systems whose inference strategies “postpone” most of the inference, necessary for this test, for the retrieval (query evaluation) phase.

The test can be executed using the **exquant-test** script (name appended by **.cmd** or **.sh**) from the **test** folder of OWLIM's installation. The generation and evaluation can be modified, using two JVM options:

- Dstepchain** sets the initial length of the chain (the default is 300) and the incremental step by which the chain grows
- Dmaxchain** determines the maximum length to be generated and tested (the default value is 600).

For each such length, an RDF test data is generated in the **test/benchmark/data** subfolder and then imported into a clean repository. The code of the test can be found in the **src\com\ontotext\trree\benchmark\TransitivityBenchmark.java** file. Then a query defined in the **test\benchmark\transitivity-benchmark-query.serql** is executed and the program reports total time used for loading the data, time for query evaluation and the number of answers returned. For the default values of the parameters, two data files are be generated, loaded and queried, one with a length of 300 and one with a length of 600 resources interconnected with a property which is transitive.

The Union test is again defined in [12] – it tests the so-called ABox¹ reasoning performance in the situation of big TBox. A large class-hierarchy is generated, by means of classes which are defined as unions of other classes, i.e. implicit subsumption. Afterwards, instances are generated for the most specific classes (the “leaves” in the class-subsumption tree). The similarity with the Exquant test is that it causes serious growth of the inferred closure (although it was not designed with this purpose), and benefits from “transitive” optimization of SwiftTRREE.

The Union test can be executed using script `union-test` (name appended by `.cmd` or `.sh`). Data generation can be controlled using three input parameters: the depth of the hierarchy, the number of classes per union (the branching-factor), and the number of instances for each leaf node. The root concept is defined to be the union of N concepts, each of which is a union of N sub-concepts until the predefined depth is reached. Then for each leaf node a number of instances are defined. The data output folder is the same as for the EXQUANT test. The query which retrieves all instances of the root concept is evaluated after each generated dataset is loaded into a clean repository. The query resides in the `benchmark/union-tree-benchmark-query.serql` file and the code of the test can be seen in the `UnionTreeBenchMark.java` file from the `src\com\ontotext\tree\benchmark` folder. The parameters cannot be controlled using runtime parameters; one must change the code of the test to check it with different initial sizes. The benchmark performs 6 tests using different Depth, union size and instances per leaf values: (5, 5, 10), (6, 5, 10), (7, 5, 10), (12, 2, 10), (13, 2, 10) and (14, 2, 10) where the number of concepts is restricted up to 3500 (35000 instances in total).

Both tests use `owlim-benchmark` repository configuration from the `test/bin/test` subfolder of the distribution.

5 Running the Wordnet Sample Application

Wordnet, <http://wordnet.princeton.edu/>, is the most popular **lexical knowledge base**, developed in the University of Princeton. It encodes the meanings of about 150 000 English words. The meanings of the words are defined by word-senses, which relate a word to a lexical concept. Lexical concepts are called *synsets*, i.e. synonym sets – about 115 000 of those appear in Wordnet v.2.0. Numerous **lexical semantic relations** are formally modeled, e.g.:

- Hyponymy (subsumption from a more-general term);
- Antonymy (negation, a term with the opposite meaning);
- Causation and entailment (for verbs).

Standard RDF/OWL representation of Wordnet is available at <http://www.w3.org/TR/wordnet-rdf/>. It contains about 1.9 million explicit statements (the Full variant), expressed in a fragment of OWL Lite. Total materialization on top of it derives another 6.3 million implicit statements.

The corresponding test application is located in the `wordnet` sub-folder of OWLIM’s distribution. To run the example, one should download the archive of the Full version from

¹ In Description Logics (DL) TBox is considered the schema part of the the knowledge base or, the ontology – the definition of classes/concepts and properties/roles. ABox is the name there for instance data – the descriptions of the individuals and the relations between them.

<http://www.w3.org/2006/03/wn/wn20/download/wn20full.zip>, extract it into a folder and provide a path to this folder as value of `preload.folder` parameter in the `owlim.properties` file. Than run the corresponding `example` script (`.cmd` or `.sh`).

One can easily modify the queries to be executed by editing file `sample-query.serq1`.

6 Running Lehigh University Benchmark (LUBM)

The LUBM benchmark is presented in [5] and briefly introduced in SwiftOWLIM's System Documentation, [10], section 9.2. The distribution includes a prebuilt library of the benchmark's source code, available for download at <http://swat.cse.lehigh.edu/projects/lubm/index.htm>. The prebuilt `lubm.jar` library is located in the `ext` distribution folder and also includes the wrapper classes that the benchmarks codebase require in order to run over Sesame repository, configured with OWLIM.

To run LUBM, it is required that the test fileset be generated beforehand. To do so one can use script named `lubm-generate` (`.cmd` or `.sh`, respectively for Windows or Linux), which is part of the distribution's sub-folder `test`. It accepts a single numeric argument `N` as the target number of the universities and creates a subfolder `univerN` where it places the generated OWL files. Once the fileset is generated, one should take a look at the `config.kb.example.owlim` file and comment/uncomment the appropriate benchmark configuration section to be executed. By default, a single University dataset is configured and its section looks as follows:

```
# 1-university
[OWLIM_1]
class=owlim.OwlimWrapper
data=./univer1
database=jdbc:ignore
ontology=http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl
```

One should use the `#` symbol at the beginning of the line to comment it, respectively, to remove this symbol to uncomment a line.

The test should be launched via the `lubm-benchmark`, script. Before either generation of datasets or execution of the test, OWLIM should be configured as discussion in section 2.

7 Running OUBM

The OUBM benchmark is introduced in [9] and briefly presented in section 9.5 of OWLIM's System Documentation, [5]. To run the test one needs to obtain the benchmark's input fileset. Because there is no fileset generator for the datasets and they were not publicly available for download from other websites, those were made available as follows:

- The DL datasets: <http://www.ontotext.com/owlim/eLUBM/eLubm-dl-apr06.zip>
- The Lite datasets: <http://www.ontotext.com/owlim/eLUBM/eLubm-lite-apr06.zip>

These archives need to be extracted in the `test` sub-folder of the OWLIM installation. To run eLUBM test OWLIM requires additional JVM parameters, used to specify the file-mask filter by which the files are selected during the upload phase of the benchmark and also a parameter to

specify the Sesame's repository identifier at which the benchmark to be performed. Those parameters are `filemask` and `testset`; sample assignments follow:

`-Dfilemask=5-ub-lite-univ` – specifying that files starting with '5-ub-lite-univ' are to be uploaded from the selected knowledge base folder (defined as a parameter in `config.kb.example` file);

`-Dtestset=eLubm-lite` – the repository id to be used for the benchmark.

The distribution comes with few Windows scripts (again in the `test` folder of OWLIM's distribution) for running those benchmarks, namely: `oubm-dl-benchmark.cmd` – for running the single-university DL benchmark; `oubm-dl-5-benchmark.cmd` – for the 5-university fileset and `oubm-dl-10-benchmark.cmd` – for the 10-university fileset; `oubm-lite-benchmark.cmd`; `oubm-lite-5-benchmark.cmd` and `oubm-lite-10-benchmark.cmd` for the Lite versions of the benchmark. The single-university scripts is given in Linux variant (named respectively `.sh`) – one can copy and modify them by hand for the larger sets.

Before running the scripts, OWLIM should be configured as discussion in section 2, and execute the scripts to perform the benchmark.

Please note that the namespaces used in 1 and 5 universities filesets are different so a special attention should be given to set those properly within the various configuration and query files. For instance the DL1 fileset uses <http://www.eclipse.org/eodm/univ-bench-dl.owl#> namespace but DL5 fileset - <http://uob.iodt.ibm.com/univ-bench-dl.owl#>.

8 References

- [1] Aduna b. v. *User Guide of Sesame*. <http://www.openrdf.org/doc/sesame/users/index.html>
- [2] Beckett, D. (editor). (2004). *RDF/XML Syntax Specification (Revised)*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/rdf-syntax-grammar/>
- [3] Carroll, J. J; De Roo, Jos. (2004). *OWL Web Ontology Language: Test Cases*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/owl-test/>
- [4] Dean, M; Schreiber, G. – editors; Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I.; McGuinness, D. L; Patel-Schneider, P. F.; Stein, L. A. (2004). *OWL Web Ontology Language Reference*. W3C Recommendation, 10 Feb. 2004. <http://www.w3.org/TR/owl-ref/>
- [5] Guo, Y; Pan, Z; and Heflin, J. (2004). *An Evaluation of Knowledge Base Systems for Large OWL Datasets*. Journal of Web Semantics, 3(2), 2005, pp158-182. <http://www.websemanticsjournal.org/ps/pub/2005-16>
- [6] Hayes, P. (2004). *RDF Semantics*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-nt-20040210/>
- [7] Kiryakov, A; Ognyanov, D; Manov, D. (2005). *OWLIM – a Pragmatic Semantic Repository for OWL*. In Proc. of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA.
- [8] Klyne, G; Carrol, J. J; (eds). (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/rdf-concepts/>
- [9] Ma, L; Yang, Y; Qiu, Z; Xie, G; Pan, Y. *Towards A Complete OWL Ontology Benchmark*. In Proc. of the 3rd European Semantic Web Conference (ESWC 2006). Budva (Montenegro).
- [10] Ontotext Lab. (2007). *SwiftOWLIM System Documentation*. v2.9.0. <http://www.ontotext.com/owlim/v2.9.0/OWLIMSysDoc.pdf>
- [11] Ontotext Lab. (2007). *BigOWLIM System Documentation*. v0.9.4. <http://www.ontotext.com/owlim/big/v0.9.4/BigOWLIMSysDoc.pdf>
- [12] Weithöner, T., Liebig, T., Luther, M., Böhm, S. (2006). *What's Wrong with OWL Benchmarks?* SSWS2006