



D2.6.3 A scalable repository for massive semantic annotation

Damyan Ognyanoff, Atanas Kiryakov, Rouslan Velkov, Milena Yankova
Ontotext Lab, Sirma Group Corp.

Abstract

Semantic annotation of unstructured content with respect to ontologies and instance data is a metadata generation task central for a wide range of Semantic Web and Knowledge Management applications. To manage semantic annotations and instance data, massive automatic semantic annotation platform needs a scalable, high-performance semantic repository with light-weight reasoning support.

OWLIM is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. In this deliverable we present BigOWLIM version which in contrast to SwiftOWLIM [18] (the „in-memory” version), performs reasoning and query evaluation directly against the permanent image of the repository. The persistence is implemented through binary files with proprietary format.

According to the publicly available results, BigOWLIM is the most scalable RDF repository with OWL reasoning support. In experiment on LUBM [8] it loaded, performed inference, stored and indexed LUBM(8000,0) in 69 hours.

Keyword list: semantics, semantic repository, ontology, RDFS, OWL, reasoning, entailment rules, Sesame

WP2 Human Language Technologies

Prototype	PU
Contractual date of delivery	M36

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.

Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Empolis GmbH

Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540
Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

Jozef Stefan Institute

Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Karlsruhe, Institute AIFB

Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592
Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Sheffield

Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK

Tel: +44 114 222 1891
Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

University of Innsbruck

Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475
Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Intelligent Software Components S.A.

Pedro de Valdivia, 10
28006
Madrid
Spain
Tel: +34 913 349 797
Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

D2.6.3 / A scalable repository for massive semantic annotation

Kea-pro GmbH

Tal
6464 Springen
Switzerland
Tel: +41 41 879 00
Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Otto-Hahn-Ring 6
81739 Munich
Germany
Contact person: Dirk Ramhorst
Tel: +49 (89)63640225; Fax: +49 89 63640233
Email: Dirk.Ramhorst@siemens.com

Ontoprise GmbH

Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912
Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Sirma Group Corp., Ontotext Lab

135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Vrije Universiteit Amsterdam (VUA)

Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Universitat Autònoma de Barcelona

Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vall`es)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Siemens Business Services GmbH & Co. OHG

Executive Summary

Semantic annotation of unstructured content with respect to ontologies and instance data is a metadata generation task central for a wide range of Semantic Web and Knowledge Management applications. A platform for massive automatic semantic annotation is presented in deliverables D2.6.1 [16] and D2.6.2, [30]. To manage the semantic annotations and the instance data, such platform needs a scalable, high-performance semantic repository with light-weight reasoning support.

BigOWLIM is a high-performance semantic repository with support for OWL reasoning and rule extensions. BigOWLIM uses the TRREE engine to perform RDFS, OWL DLP, and OWL Horst reasoning, based on forward-chaining of entailment rules. The reasoning support can be customized through rule sets. The semantics supported is customizable through rule sets. There are four pre-defined rule sets, the most expressive of which supports a proper extension of RDFS with almost full OWL Lite.

Initially, OWLIM had a single implementation (now called SwiftOWLIM), version 2.8.0 of which was described in deliverable D2.6.1, [16]. SwiftOWLIM maintains a representation of the repository contents in memory, in order to perform reasoning and query evaluation against it. Combined with its reliable persistence strategy, this turns SwiftOWLIM into fully functional semantic repository, which can scale up to 10 million statements on desktop machines and 50 million statements on a mid-range server. SwiftOWLIM is the fastest repository with OWL reasoning.

BigOWLIM is a further development, which aims at even higher scalability. In BigOWLIM, reasoning and query evaluation are performed over a storage based on binary files. The reasoning strategy is total materialization. The efficiency of TRREE allows BigOWLIM to manage billions of explicit statements on server hardware. BigOWLIM is relatively slow delete operation – a limitation typical for the OLAP databases. The upload, the inference, and the query evaluation proceed extremely fast, even against huge ontologies and knowledge bases. The version of TRREE, used in BigOWLIM, features query optimization, which assures optimal evaluation disregarding the syntactic variations of the query. Another important feature of BigOWLIM is the special handling of equality that allows for unmatched efficiency in case of datasets including large “equivalence classes”, i.e. sets of resources interconnected via `owl:sameAs`.

According to the publicly available results, BigOWLIM is the most scalable RDF repository with OWL reasoning support. In experiment on LUBM, the most popular OWL performance benchmark, it loaded, performed inference (materialization), stored and indexed LUBM(8000,0) (more than 1 billion statements) in 69 hours.

SwiftOWLIM and BigOWLIM are specific configurations of the Sesame RDF database and count on it for various sorts of features and infrastructure, including, but not limited to, an extensive set of RDF and query language parsers. BigOWLIM is packaged as a Storage and Inference Layer (SAIL) for Sesame named **BigOwlSchemaRepository**; it implements the **RdfSchemaRepository** interface. The current version of BigOWLIM is compatible with Sesame v.1.2.2-1.2.6. Further information related to various aspects of Sesame’s specification, architecture, and implementations can be found in the documentation of Sesame, [1]. BigOWLIM is commercial Java library available under a commercial license from Ontotext Lab.

Contents

SEKT Consortium	2
Executive Summary	4
Contents	5
1 Factsheet	6
1.1 Availability and Contacts	6
1.2 The Current Version Release Notes.....	6
1.3 Interfaces, Standards, Requirements.....	7
1.4 Installation and Usage.....	7
2 Introduction	8
2.1 OWL Horst: a Proper Rule Extension of RDFS.....	8
3 BigOWLIM: High-Level Architecture and Semantics	11
3.1 TRREE Engine	12
3.2 Rule Language.....	12
3.3 Supported Semantics.....	14
4 BigOWLIM versus SwiftOWLIM	18
4.1 Persistence.....	18
4.2 Indices and Query Optimization.....	19
4.3 Reasoning Strategies.....	19
5 Installation and Configuration	20
5.1 Distribution Contents	21
5.2 Configuration.....	22
5.3 Sample Memory Configurations.....	26
6 Performance Evaluation	27
6.1 LUBM Benchmark	27
6.2 LUBM(50,0) – about 7 million explicit statements.....	27
6.3 LUBM(1000,0) – above 130 million statements.....	29
6.4 LUBM(5000,0) – above 670 million statements.....	29
6.5 LUBM(8000,0) – above 1 billion statements.....	29
7 Performance Comparison	31
7.1 Up to 10 Million Statements (Reasoners).....	31
7.2 Above 10 Million Statements (Repositories)	34
Bibliography and references	36

1 Factsheet

OWLIM is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM uses the TRREE engine to perform RDFS and OWL reasoning, based on forward-chaining of entailment rules. The semantics supported can be customized through selection of one of the predefined rulesets or definition of a proprietary one. The most expressive predefined ruleset supports a proper extension of RDFS with almost full OWL Lite.

BigOWLIM performs reasoning and query evaluation directly against the permanent image of the repository. The persistence is implemented through binary files with proprietary format. BigOWLIM is relatively slow on delete – a limitation typical for the OLAP databases. The upload, the inference, and the query evaluation proceed extremely fast, even against huge ontologies and knowledge bases. The version of TRREE, used in BigOWLIM, features query optimization, which assures optimal evaluation disregarding the syntactic variations of the query. Another important feature of BigOWLIM is the special handling of equality that allows for unmatched efficiency in case of datasets including large “equivalence classes”, i.e. sets of resources interconnected via `owl:sameAs`.

According to the publicly available results, BigOWLIM is the most scalable RDF repository with OWL reasoning support. In experiment on LUBM, the most popular OWL performance benchmark, it loaded more than 1 billion statements and performed inference on top of them. To be more precise, BigOWLIM loaded LUBM(8000,0) and answered the queries in 69 hours.

1.1 Availability and Contacts

Version: 0.9.2-beta, pre-release, 4 Oct. 2006.

Download: not available for free download.

Contact: mailing lists available at <http://www.ontotext.com/owlim>.

1.2 The Current Version Release Notes

- **Custom inference**: the TRREE’s rule compiler became part of the distribution, which allows the usage of custom rule-sets for inference. This way one can specify semantics which best fits the concrete application in terms of expressivity and performance;
- **Command line parameters**: many of the parameters of OWLIM can now be passed through the command line. In the previous versions, those could have only been set as SAIL parameters in the `system.conf` file or programmatically.
- **Minor fixes in the owl-max rule-set**: those allow for covering some extra cases of A-Box reasoning and eliminate most of the cases when B-Nodes have been generated.

- **Linux shell scripts:** Linux scripts have been added to the distribution, which allow for control (start/stop) of a standalone version of OWLIM and running tests. Such scripts were available only for Windows in previous versions.
- **Query evaluation fixes:** few problems, related to proper handling of Sesame construct queries, were fixed in BigTRREE; those were detected after a bug report from a user;
- **Equivalence classes support fixes:** some owl:sameAs statements were not properly inferred in the previous version;
- **Initialization from file images fixed:** some bugs related to the generation of B-Nodes and some of the in-memory structures were fixed;
- **Temporary file creation fixed:** improper handling of relative storage folder name was causing problems with temporary file creation.

1.3 Interfaces, Standards, Requirements

- **Nature:** Java library without user interface.
- **Interfaces (API, Web Services):** Java API, RMI.
- **Platform:** JDK 1.4.2 and 1.5 (both 32-bit and 64-bit versions tested).
- **Supported standards:** BigOWLIM is bound to the data and query standards supported by Sesame. RDF is the basic data standard, [14]; the supported query languages are: SeRQL, RQL, RDQL.
- **Syntaxes:** The import and export of all major RDF syntaxes (XML, N3, N-Triples) is supported by Sesame.
- **Semantics:** BigOWLIM supports RDFS, OWL DLP, OWL Horst (an OWL dialect more expressive than DLP and backward compatible with RDFS), and partially OWL Lite.
- **Required Libraries:**
 - Sesame (<http://www.openrdf.org>) is an open-source RDF database with a support for RDF inference and querying. OWLIM is a SAIL that implements the `RDFSRepository` interface. Big OWLIM v0.9.2-beta was tested with Sesame releases 1.2.1-1.2.6.
 - TRREE (<http://www.ontotext.com/tree/>) is a Triple Reasoning and Rule Entailment Engine, which allows forward-chaining and materialization with respect to entailment rules. Within Big OWLIM, the version of TRREE comes preconfigured with four distinct sets of rules.

1.4 Installation and Usage

BigOWLIM is distributed as a ZIP archive that contains the `owlim-big-0.9.2-beta.jar`, all the required libraries, configuration files, sources, documentation, and sample data. BigOWLIM is designed to be used as a Storage and Inference Layer (SAIL) of Sesame – no BigOWLIM specific public interfaces exist. Since Sesame is not included in the BigOWLIM distribution, it should be downloaded from <http://www.openrdf.org>.

2 Introduction

For the purpose of this presentation we refer to semantic annotation as (i) a sort of meta-data and since we are interested in repositories we will not discuss (ii) the process of generation of such meta-data here. So, once the semantic annotation of a text is performed, it contains information about what entities appear in a text and where they do. The result is formally recorded and associated with the place in the text where the entity has been mentioned. The identity of the entity is "verbalized" via URIs which means that those can be easily linked to their descriptions within a semantic repository. Thus, semantic annotations are references from the text to a semantic repository, containing further knowledge.

Semantic annotation of unstructured content with respect to ontologies and instance data is a metadata generation task central for a wide range of Semantic Web and Knowledge Management applications. A platform for massive automatic semantic annotation is presented in deliverables D2.6.1 [16] and D2.6.2, [30]. To manage the semantic annotations and the instance data, such platform needs a scalable, high-performance semantic repository with light-weight reasoning support.

The focus of this deliverable is storage of semantic annotations and we present OWLIM – a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM uses the TRREE engine to perform RDFS and OWL reasoning, based on forward-chaining of entailment rules. The semantics supported can be customized through selection of one of the predefined rulesets or definition of a proprietary one. The most expressive predefined ruleset supports a proper extension of RDFS with almost full OWL Lite.

This section continues with introductory discussion on various fragments of OWL and in particular tractable fragments like OWL DLP and OWL Horst/Tiny. Section 3 covers the major design issues, including reasoning and supported semantics. Section 4 compares BigOWLIM and SwiftOWLIM (the „in-memory” version of OWLIM), thus providing better understanding of the behaviour of both engines. Section 5 presents the most significant installation and configuration issues. The later is important, as long as there is no single optimal configuration with all semantics, datasets, hardware configurations, usage patterns. Section 6 reports the results of the performance evaluation and concludes with comparison to other engines.

2.1 OWL Horst: a Proper Rule Extension of RDFS

In [24] ter Horst defines RDFS extensions towards rule support and describes a fragment of OWL, more expressive than DLP. He introduces the notion of R-entailment of one (target) RDF graph from another (source) RDF graph on the basis of a set of entailment rules R . R-entailment is more general than the D-entailment used by Hayes, [9], in defining the standard RDFS semantics. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are “extended” RDF statements, where variables can take any of the three positions. The head of the rule comprises of one or more consequences, each of which is, again, an extended RDF statement. The consequences may not contain free variables, i.e. which are not used in the body of the rule. The consequences may contain blank nodes.

The extension of the R-entailment (as compared to the D-entailment) is that it “operates” on top of the so-called generalized RDF graphs, where blank nodes can appear as predicates. R-entailment rules without premises are used to declare axiomatic statements. Rules without consequences are used to imply inconsistency.

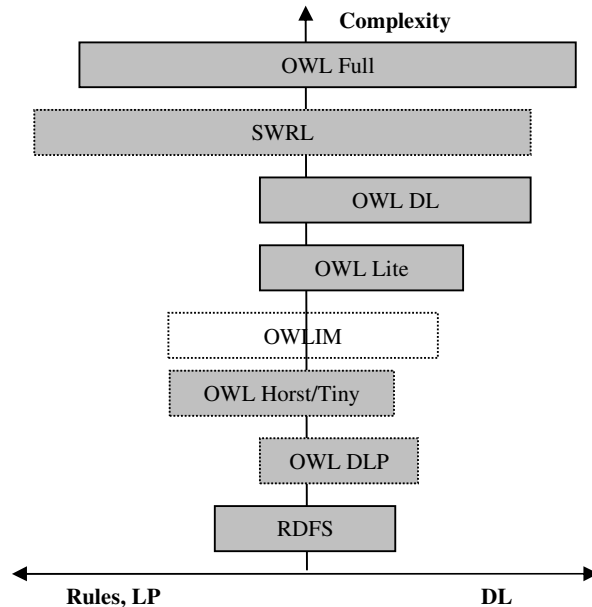


Figure 1. OWL-Related Languages Map

Horst extends and modifies the D-entailment rules from [9] in two steps as follows: D* adds entailment support for literal datatypes; pD* adds rules which provide partial support for OWL. The primitives involved are the following: `FunctionalProperty`, `InverseFunctionalProperty`, `SymmetricProperty`, `TransitiveProperty`, `sameAs`, `inverseOf`, `equivalentClass`, `equivalentProperty`, `onProperty`, `hasValue`, `someValuesFrom`, `allValuesFrom`, `differentFrom`, `disjointWith`.

The last two primitives are supported through inconsistency rules which fire in case of the so-called P-clashes. It is important to acknowledge that some of the primitives are only partially supported; the standard OWL entailments related to `someValuesFrom` and `allValuesFrom` are supported only in one of the directions (i.e. there is no full support for the iff-semantics of these OWL primitives).

We refer to this extension of RDFS as “OWL Horst”. As outlined in [24], this language has a number of important characteristics:

- It is a proper (backward-compatible) extension of RDFS. In contrast to OWL DLP, it puts no constraints on the RDFS semantics. The widely discussed meta-classes (classes as instances of other classes) are not disallowed in OWL Horst. It also does not enforce unique name assumption;
- Unlike the DL-based rule languages, like SWRL, [10] and [17], R-entailment provides a formalism for rule extensions without DL-related constraints;

- Its complexity is lower than the one of SWRL and other approaches combining DL ontologies with rules; see section 5 of [24].

Figure 1 presents a simplified map of the complexity of a number of OWL-related languages, as well as their bias towards DL and LP-based semantics. An extended discussion on the topic can be found at: http://www.ontotext.com/inference/rdfs_rules_owl.html. The “OWLIM” box depicts the position on the map of the most complex OWL dialect supported by OWLIM – rule-set **owl-max** with **partialRDFS** parameter set to **false**; see section 7.3. of [18] for parameter description. The pre-defined rule sets in OWLIM do not support entailment of typed literals (D-entailment); more details on the semantics supported by OWLIM can be also found in section 3.3.

OWL Horst is close to what has been intuitively described as *OWL Tiny* at the SWAD-Europe¹ workshop in VU Amsterdam, Nov 2003. The major difference is that OWL Tiny (similarly to the fragment supported by OWLIM) does not support entailment over data types.

¹ As described at the SWAD-Europe Workshop on Semantic Web Storage and Retrieval, Amsterdam, Nov 2003, http://www.w3.org/2001/sw/Europe/reports/dev_workshop_report_4/#owl-tiny.

3 BigOWLIM: High-Level Architecture and Semantics

BigOWLIM is a specific configuration for the Sesame RDF database and counts on it for various sorts of features and infrastructure, including, but not limited to, an extensive set of RDF and query language parsers. BigOWLIM is “packaged” as a Storage and Inference Layer (SAIL) for Sesame named `BigOwlImSchemaRepository`; it implements the `RdfSchemaRepository` interface. The current version of BigOWLIM is compatible with Sesame v.1.2.2-1.2.6. Further information related to various aspects of Sesame’s specification, architecture, and implementations can be found in the documentation of Sesame, [1].

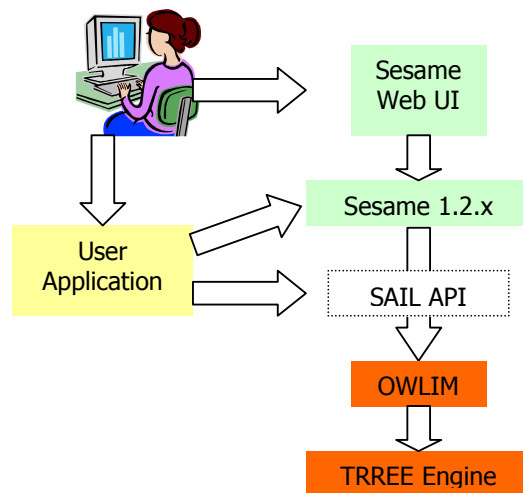


Figure 2. OWLIM Usage and Relations to Sesame and TRREE

As presented on Figure 2, OWLIM implements the SAIL interface, so, it is integrated with all the facilities of Sesame, e.g. the query engines and the web UI. A user application can choose whether to use OWLIM directly, through the low-level SAIL interface, or through the higher-level functional interfaces.

In BigOWLIM, reasoning and query evaluation are performed over a storage based on binary files. The reasoning strategy is total materialization. OWLIM uses the TRREE engine to perform RDFS and OWL reasoning, based on forward-chaining of entailment rules. The efficiency of TRREE allows BigOWLIM to manage billions of explicit statements on server hardware. BigOWLIM is relatively slow delete operation – a limitation typical for the OLAP databases. The upload, storage, inference, and query evaluation are fast even for huge ontologies and knowledge bases.

3.1 TRREE Engine

Like the SwiftOWLIM version [18] of OWLIM, BigOWLIM uses TRREE² (Triple Reasoning and Rule Entailment Engine). It is implemented in Java and its early versions were named IRRE. TRREE performs reasoning based on forward-chaining of entailment rules over RDF triple patterns with variables. The rule format and the semantics enforced is analogous to those of the R-entailment (see [24]) with the following differences:

- Free variables in the head (without binding in the body) are considered as blank nodes. This feature can be considered as “syntactic sugar”;
- Variable inequality constraints can be specified in the body of the rules, in addition to the triple patterns. While these constraints do not change the complexity of the entailment, they allow for more efficient entailment; one can consider them as hints for the reasoner;
- `[cut]` operator can be associated with rule premises, the TRREE compiler interprets it like the `!` operator in Prolog;
- Inconsistency rules are not supported, i.e. there is no specific mechanism to allow inconsistency checks. One can easily model these via regular rules which entail `<x, rdf:type, owl:Nothing>` statements, without affecting the complexity class;
- Axioms can be provided as a set of statements, although, those are not modelled as rules with empty bodies.

TRREE can be configured via “rulesets” – sets of axiomatic triples and entailment rules, which determine the supported semantics. The implementation of TRREE relies on a compile stage, during which the rules are compiled into chunks of Java code that are post-processed and merged together to generate the main entry point for the inferencer.

3.2 Rule Language

The ruleset format supported by TRREE is derived from the one of the Custom Inferencer of Sesame. Formally, a ruleset file should consist of exactly three sections enclosed in curly braces ‘{’ and ‘}’ named **Prefices**, **Axioms** and **Rules** which (if present) must appear in exactly this ordering. Only the **Rules** section is mandatory, the other two are optional.

The most common way to include comments is taken, where ‘/*’ and ‘*/’ are used to enclose a multi-line comment and ‘//’ for a single line (contents until the end of the line are ignored).

The **Prefices** section is the place where the prefixes of the common namespaces are defined. Those could be used later on in **Axioms** and **Rules** sections for qualifying resources with their local name only. Each pair prefix/namespace should occur on a separate line in the file, as in the following example:

² <http://www.ontotext.com/trree/>

Prefices

```
{
  rdf      : http://www.w3.org/1999/02/22-rdf-syntax-ns#
  rdfs     : http://www.w3.org/2000/01/rdf-schema#
  owl    : http://www.w3.org/2002/07/owl#
  protons  : http://proton.semanticweb.org/2005/04/protons#
  protont  : http://proton.semanticweb.org/2005/04/protont#
  protonu  : http://proton.semanticweb.org/2005/04/protonu#
  protonkm : http://proton.semanticweb.org/2005/04/protonkm#
  xsd     : http://www.w3.org/2001/XMLSchema#
}
```

Note: Please, avoid defining a namespace using the `http` prefix since it will be used to expand the node URIs in a weird way.

The **Axioms** section encloses a set of axiomatic triples to be asserted initially into the repository. Those triples are usually used to describe the meta-level primitives used to define the schema, such as `rdf:type`, `rdfs:Class`, etc. Each line of the section should consist of exactly three node names, enclosed in ‘<’ and ‘>’ symbols and delimited by a white-space. Each line defines a single axiomatic triple. Here follows an example **Axiom** section defining part of the RDFS axiomatic triples:

Axioms

```
{
// RDF axiomatic triples :

  <rdf:type> <rdf:type> <rdf:Property>
  <rdf:subject> <rdf:type> <rdf:Property>
  <rdf:predicate> <rdf:type> <rdf:Property>
  <rdf:object> <rdf:type> <rdf:Property>
  <rdf:first> <rdf:type> <rdf:Property>
  <rdf:rest> <rdf:type> <rdf:Property>
  <rdf:value> <rdf:type> <rdf:Property>
  <rdf:nil> <rdf:type> <rdf:List>

// RDFS axiomatic triples :

  <rdf:type> <rdfs:domain> <rdfs:Resource>

  <rdfs:domain> <rdfs:domain> <rdf:Property>
  <rdfs:range> <rdfs:domain> <rdf:Property>
}
```

The **Rules** section encloses all the rule definitions. Every rule starts with a mandatory rule identifier (ID), which should appear in the beginning of a new line just after the **Id:** string. The rules are defined via statement patterns – one or more premises and one or more corollaries in the following form:

```
Id: Rule_Id
  < Premise #1 >
  < Premise #2 >
  . . .
  < Premise #n >
-----
  < Corollary #1 >
  < Corollary #2 >
  . . .
  < Corollary #m >
```

The premises and corollaries are defined via a subject, predicate, and object components, each of which could be as follows:

- A variable, represented as a single Latin letter; or
- A full URI or its short name, formed by a prefix as it was defined in the **Prefices** section, followed by ‘:’ and its local name. Each URI should be enclosed in ‘<’ and ‘>’ symbols; or
- A literal, in its standard XML encoding.

Additionally, each premise may contain one or more constraints, delimited by comma and enclosed in ‘[’ and ‘]’ symbols, stating that the value of one or more variables in the statement must not be equal to some specific URI or to the value that is bound to another variable from the same rule. Each constraint expression should start with the **Constraint** constant. Here follows an example of a rule:

```
Id: test_rule
  a <myns:myproperty> b      [Constraint a != b]
  <myns:Instance_1.0> a c  [Constraint a!=<rdf:type>, c!=a, c!=b]
  -----
  c a "3"^^xsd:nonNegativeInteger
  b <rdf:type> <myns:Instance_1.0> [Constraint b != <rdfs:Class>]
```

The LHS value in the not-equal constraint must denote a variable and the RHS value can be a variable, a full URI, or a short form of such. Inequality constraints may be used to force the engine not to apply the rule when the constraints are not satisfied, which will improve engine's performance.

Constraints are valid within the line they appear. If a variable is not bound yet then the constraint is considered satisfied (and therefore does not apply).

The premises and corollaries are delimited by a single line consisting of one or more ‘-‘ symbols (only this symbol is permitted for such a delimiter line).

3.3 Supported Semantics

The semantics supported by OWLIM is configurable – the TRREE engine can be configured to work with one out of a set of several pre-defined rulesets. Those are properly “nested” in each other; a list ordered by increasing complexity is given below:

- **empty**: no sort of reasoning, i.e. OWLIM acts as a plain RDF store;
- **rdfs**: support for the standard model theoretic RDFS semantics;
- **owl-horst**: OWL dialect close to OWL Horst; the differences are discussed below;
- **owl-max**: a combination of most of the semantics of OWL Lite in combination with full compatibility with (support for) RDFS.

The so called **partialRDFS** modification of the predefined rulesets and how one can define custom rulesets is described in more details in section 6 of [18]. The ruleset to be used for specific repository is defined through the **ruleset** parameter.

The richest ruleset (**owl-max**) encompasses the model theoretic semantics of RDFS, as defined in [9], extended with support for most of the OWL primitives, as follows: **SymmetricProperty**, **TransitiveProperty**, **inverseOf**, **equivalentClass**, **equivalentProperty**, **sameAs**, **FunctionalProperty**, **InverseFunctionalProperty**, **onProperty**, **allValuesFrom**, **someValuesFrom**, **hasValue**, **unionOf**, **intersectionOf**, **differentFrom**, **oneOf**, **AllDifferent**, **minCardinality**, **maxCardinality**, **cardinality**. The major limitations for the support of these primitives are as follows:

- Cardinality constraints in the range 0-1, as in OWL Lite;
- The support of **allValuesFrom**, **someValuesFrom**, **hasValue** and **unionOf** includes rules that “close” the semantics in one direction only. See the corresponding rules for details;
- **differentFrom** statements are generated exclusively from **AllDifferent**, but no inference is carried out based on them. In case that two RDF nodes are linked through both **sameAs** and **differentFrom** properties, inconsistency checking is left to the application;
- Cardinality constraints are handled if the constraint values are given as XML literals of type **xsd:nonNegativeInteger**; literals typed as **xsd:integer** are not properly handled.

Regarding OWL compliance, OWLIM supports a dialect (ruleset **owl-horst**) similar to OWL Horst, as defined in [24] and introduced in section 2.1. The following statements can be made for the OWL support in OWLIM (considering the **owl-max** ruleset, without **partialRDFS** optimizations):

- OWLIM supports the full RDFS semantics without constraints or limitations, apart from the entailment with typed literals (D-entailment). For instance, meta-classes (and any arbitrary mixture of class, property, and individual) can be combined with the supported OWL semantics;
- The supported OWL semantics does not cover the full OWL Lite. For instance, the support for **owl:minCardinality** is incomplete even in the case when the constraint is set to 1. Still, it has to be considered that the OWL semantics supported by OWLIM, combined with unconstrained RDFS semantics, constitutes a language which is not comparable to OWL Lite – its full compatibility with RDFS makes it more expressive than OWL Lite in some aspects;
- OWLIM supports a language richer than OWL DLP;

The differences between the OWL dialect supported through the **owl-horst** ruleset by OWLIM and OWL Horst can be summarized as follows:

- OWLIM does not provide the extended support for typed literals, introduced with the D*-entailment extension of the RDFS semantics. Although such support is conceptually clear, it is our understanding that the performance “penalty” is too high for most of the applications;

- OWLIM is not a DL reasoner by design, which considering the theoretical constraints against a full DL reasoner, we consider a major advantage.
- There are no inconsistency rules, as those defined in [24] for `disjointWith` and `differentFrom`.
- Few more OWL primitives are supported by OWLIM (ruleset `owl-max`), namely: `unionOf`, `intersectionOf`, `AllDifferent`, `oneOf`, `minCardinality`, `maxCardinality`, `cardinality`.
- There is extended support for schema-level (T-Box) reasoning in OWLIM. Such reasoning is implemented for instance with `owl_subClassBetweenSomeVal` and `owl_FunctPropByInvFunc` rules.

Even though the concrete rules, pre-defined in OWLIM, differ from the ones defined in OWL Horst, in [24], the complexity and decidability results reported for R-entailment are relevant for TRREE and OWLIM. To be more precise, rules in the `owl-horst` ruleset, do not introduce new B-Nodes, which means that R-entailment with respect to them takes polynomial time. In KR terms, this means that the `owl-horst` inference within OWLIM is tractable. Further, in the `owl-max` ruleset, there is a single rule that introduces B-nodes, as follows:

```
Id: owl_minMaxCardByCard
r <owl:onProperty> p
r <owl:cardinality> y
-----
w <owl:onProperty> p
w <owl:minCardinality> y
z <owl:onProperty> p
z <owl:maxCardinality> y
r <rdfs:subClassOf> w
r <rdfs:subClassOf> z
```

Analysis of the ruleset shows, that this rule can be removed, at the cost of introducing a specific one for cardinality 1 and duplicating the rules which “consume” min-/max-cardinality-1 constraints. So, it shall not have impact on the reasoning complexity in real-world ontologies.

This provides evidence that OWLIM implements reasoning with lesser complexity, as compared to other formalisms, which combine DL ontologies with rules. In addition, it puts no constraints with respect to meta-modelling.

The correctness of the support of the OWL semantics (for those primitives which are supported) is checked against the normative Positive and Negative-entailment OWL test cases, [4]. These checks are available as JUnit tests together with the OWLIM distribution; they are documented in [18] and can also be used as sample applications.

The concrete axioms (axiomatic statements) and rules, defining the semantics supported by OWLIM, can be found in file `rules.pie` which is part of the distribution. OWLIM allows customization of the supported semantics – there are several predefined levels of entailment. One of those should be selected and specified (through the `ruleset` parameter, see section 5.2 in [18]) for each specific repository instance. Applications, which do not need the complexity of the most expressive semantics supported, can choose one of the lower levels, which will result in faster

D2.6.3 / A scalable repository for massive semantic annotation

inference. The separation of the rules into the different levels is clearly indicated (through comments) in the file `rules.pie`.

Generally, the semantics and the inference strategies supported by BigOWLIM are the same as those of SwiftOWLIM. The differences, discussed in section 4, affect only the performance of the versions of the TRREE engine, while the formal inference capabilities are equivalent.

4 BigOWLIM versus SwiftOWLIM

BigOWLIM can be considered as an “enterprise” version of SwiftOWLIM – the “standard” free OWLIM repository. Both versions are based on the TRREE engine, however, they use different versions of it, which are named according to the same convention: BigTRREE and SwiftTRREE. Table 1 represents a summary of the differences between the two versions of OWLIM; many of those differences are actually differences between the different versions of TRREE. The consequent subsections comment on the principle distinctions.

Table 1 SwiftOWLIM-to-BigOWLIM Comparison

	SwiftOWLIM	BigOWLIM
Scale (Mill. of explicit statem.)	10 MSt, using 1.6 GB RAM 40 MSt , using 6 GB RAM	130 MSt, using 1.6GB 1068 MSt , using 12GB
Processing speed (load+infer+store)	25 KSt/s on notebook 150 KSt/s on server	3 KSt/s on notebook 20 KSt/s on server
Query optimization	No	Yes
Persistence	Back-up in N-Triples	Binary files, allowing instant initialization
Efficient owl:sameAs	No	Yes
Licence and Availability	Open-source under LGPL; Uses SwiftTRREE that is free, but not open-source	Commercial Evaluation copies provided on request

4.1 Persistence

Both versions assure persistence of the stored RDF/OWL data and both of them use various sorts of in-memory tables and indices. The major difference is that BigOWLIM uses the permanent storage files for reasoning and query answering, while SwiftOWLIM uses them only as a sort of a back up. To operate SwiftOWLIM needs to have all the data in the main memory – it does not access the permanent storage files in the course of query evaluation. Here follow few of the functional differences, caused by this difference in the architectures:

- BigOWLIM scales much better, because only some key data need to be kept in memory. Still, configurable caching strategy allows Big to take advantage of the available memory for the sake of better performances (see section 7).
- BigOWLIM performs startup and initialization of large repositories much faster, because it does not need to parse, re-load and re-infer all the knowledge from scratch.
- BigOWLIM performs specific merge operation on regular basis. Those are necessary in order to keep up to date the primary indices, so to provide an optimal inference performance during upload. The frequency at which those operations are initiated is directly related to the size of the triple cache defined in the repository configuration.

- The current release of BigOWLIM, require that the primary indices should be up to date before the repository can handle queries. This limitation will affect the time for evaluating the first query posed against the repository after modifying its contents. This issue will be resolved for the next release.

4.2 Indices and Query Optimization

The indices of the SwiftOWLIM are in essence in-memory hash tables. The indices of BigOWLIM can be seen as ordered lists, which are stored permanently. BigOWLIM is also collecting various statistics about the data. The combination of its ordered indices and statistics allows BigOWLIM to perform query planning and optimization similar to this implemented in the relational databases. The lack of such planning in the SwiftOWLIM is the reason why a change in the order of the constraints (e.g. triple patterns) in a query can lead to considerable changes in the evaluation time.

4.3 Reasoning Strategies

The major difference between the reasoning capabilities of the two versions is the special support for equivalent resources. A special strategy is implemented which allows for efficient handling of datasets with heavy usage of `owl:sameAs`. The impact of this strategy gets bigger as the classes of equivalence³ are getting bigger.

Another difference is that BigOWLIM always works in the so-called stack-safe mode of the TRREE engine, which makes sure that stack overflows cannot happen during reasoning - please, refer to the description of the `stackSafe` parameter of SwiftOWLIM, in [18].

³ We call "equivalence class" the clusters of RDF nodes interconnected with `owl:sameAs` property

5 Installation and Configuration

OWLIM is a plug-in, namely a Storage and Inference Layer (SAIL), for the Sesame RDF database. To configure, run, and use it, means to do so for a specific configuration of Sesame. Please refer to the online documentation of Sesame, [1], for detailed information regarding its installation, configuration, and interfaces.

An application can use OWLIM in two modes:

- **Embedded mode:** as a library, invoked in the same process as the application, that uses it.
- **Remote access:** running as a standalone server in a separate process (possibly on a different machine); in this case the communication with the application proceeds through RMI calls.

In either case, the OWLIM's and Sesame's JAR files (Java libraries) should be included in the class-path of the application. Please, pay attention to the following:

- The OWLIM's JAR is as follows: `owlim-big-0.9.2beta.jar`. It can be found in the `lib` subfolder of OWLIM's distribution. The BigTRREE engine is part of this JAR;
- The Sesame's JARs are as follows: `openrdf-model.jar`, `openrdf-util.jar`, `rio.jar`, `sesame.jar`. One should download a Sesame distribution (v.1.2.1-1.2.6) from <http://www.openrdf.org>.
- The OWLIM's JAR must be placed in the class-path before those of Sesame.
- The usage of the rule-compiler (i.e. custom rulesets) requires `tools.jar` from the JDK to be also included in the class-path.

A sample class-path, for an application using OWLIM, may look as follows:

```
owlim-big-0.9.2beta.jar;openrdf-model.jar;openrdf-
util.jar;rio.jar;sesame.jar;tools.jar
```

Instructions and samples of how an application uses Sesame in embedded mode can be found in Sesame system documentation, [1]. In short, one should obtain a `SesameService` instance, then a repository, as demonstrated in the code snippet below:

```
// Example: initialize, using an external configuration file:
File sysConf = new File("./sesame.conf");
LocalService ss = Sesame.getService(sysConf);

ss.login("admin", "<REPLACE_ME>");

// get a local repository
LocalRepository rep = (LocalRepository)ss.getRepository("owlim");

// issue a query
QueryResultsTable result =
    rep.performTableQuery(QueryLanguage.SERQL,
        "select * from {X} rdf:type {rdfs:Class}");
// just dump the results
for (int i = 0; i < result.getRowCount(); i++) {
    System.out.println(result.getValue(i, 0));
}
```

D2.6.3 / A scalable repository for massive semantic annotation

```
// shutdown the local service
ss.shutdown();
```

To enable Sesame to close properly, one should shutdown the service, as shown above. Even without shutdown, or in case of abnormal termination, OWLIM's persistence strategy guarantees that there will be no loss of information or inconsistencies in the repository. Still, it is recommended that the Sesame shutdown mechanism is used, since it affects the initialization time for the next run.

The remote use of a standalone OWLIM server through RMI is almost transparent – some differences occur in the way of obtaining the `SesameService` instance at the beginning, as it is explained in section 7.2 of SwiftOWLIM's documentation, [18]. One can find there also instructions on starting and stopping a standalone OWLIM with `owlim_control` script, which is part of the distribution.

To a major extent, the installation and configuration of BigOWLIM are equivalent to those for SwiftOWLIM, [18]. The differences are related to the parameters related to persistency and caching.

5.1 Distribution Contents

The distribution of BigOWLIM includes:

- `lib` folder: contains BigOWLIM as a JAR file (Java library), together with all the necessary third party libraries (except those of Sesame).
- `doc` folder: system documentation and test documentation. The System Documentation of SwiftOWLIM is also included, as long as provides complementary information;
- `test` folder: contains sources, data and documentation of the unit tests and benchmarks documented in [19]. The three major sample applications are:
 - The LUBM benchmark framework, [8]; eLUBM, [15], can also be evaluated;
 - OWLIM's JUnit-based regression tests for inference;
 - An implementation of the normative Positive and Negative-entailment OWL test cases, [4].
- `owlim_control` script (`.cmd` or `.sh`, respectively for Windows or Linux) file: a batch file, allowing start and stop of OWLIM as a standalone server; see section 7.2 of [18].
- `bigowlim-mem-calc.xls` – an Excel sheet to ease the tuning of the size of the various caches and the amount of memory that BigOWLIM will use to process the input data. Please refer to section 5.3 for details;
- `*.pie` files: contains the predefined rule sets, which determine the semantics of BigOWLIM. As discussed in sections 3.1 and 6 of [18], those can be used a starting point for design of custom rule sets.

5.2 Configuration

Sesame is configured in an XML configuration file, which can be provided also at the stage of retrieving the `SesameService` instance. Follow the instructions on how to set up the related parameters in the configuration file.

The SAIL can be initialized with a set of schema files. It should also include the original OWL XML/RDFS file (`owl.rdfs`) – a copy of this file is included in this distribution for convenience.

The BigOWLIM configuration parameters, with their default and allowed values, are listed in Table 2 together with a short description of each parameter.

In this version, many of the OWLIM's specific configuration parameters could be set via JVM system properties, passed as command line parameters when starting JVM. Notes are provided below to indicate which parameters cannot be passed through the command lines; this is typically because their values may contain line separator characters. The values of configuration parameters set through the command line override any of the values present in the repository configuration (`system.conf`) file or the defaults, in case a parameter value was not set in the repository configuration. For instance, a `ruleset` parameter could be set at command line by adding the following command line argument when JVM is launched:

```
-Druleset=owl-max
```

Table 2 BigOWLIM Configuration Parameters

Name	Default	Allowed Values
<code>ruleset</code>	<code>owl-horst</code>	<code>owl-max, owl-horst, rdfs, empty</code>
	<p>Specifies the set of axioms and entailment rules used for inference, which determines the supported semantics. OWLIM is packaged with four preconfigured sets, whose names are valid values of this parameter:</p> <ul style="list-style-type: none"> owl-max including all the rules and axioms included in the <code>rules.pie</code> file. This setting provides support for OWL Lite (with some limitations) and RDFS. See [18] for a detailed explanation; owl-horst provides support for RDFS and an OWL dialect referred here as OWL Horst; rdfs including only the RDFS entailment rules equivalent to the ones given in [9]; the rules presented in section 6 of [18] are not included, i.e. there is no reasoning support for the OWL primitives; empty any sort of entailment is switched off; with this setting OWLIM functions as a plain RDF store without inference. <p>For details on the different rule-sets, refer to section 6 of [18].</p>	

	<p>Example:</p> <pre><param name="ruleset" value="owl-max"/></pre> <p>Note: if the value of this parameter does not match any of the above mentioned – it is considered to be a filename of a custom rule set. The ‘.pie’ extension is appended, if not present. Then, if such file exists, it is processed, compiled and loaded dynamically so the TRREE engine could perform inference based on the rules defined there. Regarding custom rule sets, please refer to section 6.3 of [18]</p>	
partialRDFS	true	false, true
	<p>This parameter switches on (when set to true) and off the optimization of the RDFS inference described in section 6 of [18]. Its purpose is to suspend part of the RDFS entailments, which are useless for many datasets and applications, but require considerable reasoning resources. In essence, suspended is the entailment of statements of the sort <code><?X, rdf:type, rdfs:Resource></code>, as well as such inferring membership to <code>rdfs:Class</code> and <code>rdf:Property</code>.</p> <p>This optimization has no effect, when the rule set parameter is set to empty; it has the same effect for all the other rule-sets.</p>	
storage-folder	./owlim-storage	Absolute or relative folder name which BigOWLIM will use to store the indexes.
	<p>Sets the name of folder that will be used for storing the indexes and additional data. If the trailing path separator is missing from the end of folder name, it is appended. Example:</p> <pre><param name=" storage-folder" value="/usr/shared/owlim"/></pre>	
baseURL	No default value	Any valid URL
	<p>Specifies the default namespace for the persist file. Non-empty namespaces are recommended, because their use guarantees the uniqueness of the anonymous nodes that may appear within the repository.</p> <pre><param name="base-URL" value="http://www.ontotext.com/kim/2004/12/wkb#"/></pre> <p>Note: This parameter cannot be set via command line argument</p>	
imports	No default value	Semicolon-delimited list of file names
	<p>A list of schema files which will be imported – all the statements, found in these files, will be loaded in the repository. The current version DOES NOT treat these statements as read-only and the application could remove any of them. I case the repository is</p>	

	<p>completely cleared, the contents of the files will be re-asserted at the end of the transaction. The serialization format is assumed to be RDFS/XML, unless the file has a .NT extension. Example:</p> <pre><param name="imports" value="owl.rdfs;protons.owl;protont.owl"/></pre> <p>Note: This parameter cannot be set via command line argument</p>	
defaultNS	None	Semicolon-delimited list of URLs. The number and order should match this in the imports parameter.
	<p>Specifies the default namespaces for each of the imported files (see imports parameter). Example:</p> <pre><param name="defaultNS" value="namespaces list for the imported files" /></pre> <p>Note: This parameter cannot be set via command line argument</p>	
page-cache	10000	A positive Integer number
	<p>Defines the number of pages to be cached for speeding the file operations. This number is given per-index; currently the BigOWLIM uses two separate indexes, so the total number of pages will be a twice larger. The size of a single page is 16000 bytes. As a recommendation, the value of this parameter should be at least 50 to allow for efficient file operations.</p> <pre><param name="page-cache" value="1000" /></pre>	
entity-index-size	1000000	A positive integer number
	<p>Defines the size of the index used to access the Entities from the dictionary. The dictionary includes all unique resources, blank nodes and literals found during the work of the repository. As a recommendation, the value of this parameter should be somewhere between a half and twice the number of expected entities. A smaller value will affect the upload performance of the repository.</p> <pre><param name="entity-index-size" value="1000000" /></pre>	
triple-cache-size	4000000	A positive integer number
	<p>Defines the size in triples of the statements kept in-memory. The value of this parameter will affect the upload performance. When this cache gets full a re-indexing is initiated and both primary indexes are recomputed. So smaller values will result of more frequent re-indexing operations which could become time consuming as the repository grows in size. The re-indexing operation is linear over the volume of data.</p>	

	<code><param name="triple-cache-size" value="4000000" /></code>	
key-index-size	1000000	A positive integer number
	<p>Defines the index size for the cached portion of the repository (see <i>triple-cache-size</i> parameter). As a recommendation, the value of the parameter should be close to the half of the triple-cache-size to allow for efficient upload performance.</p> <p><code><param name="key-index-size" value="1000000" /></code></p>	
cardinality-datatype	xsd:integer	An URI value
	<p>This parameter specifies the data type of the literal constants used in the cardinality restrictions. Since the BigOWLIM doesn't derive equality between literals based on the subsumption of their data types, this data type will be reused within the rules that deal with cardinality. For instance the <code>"1"^^xsd:integer</code> and <code>"1"^^xsd:nonNegativeInteger</code> are distinct literals and the specific rules about restrictions for <code>minCardinality</code> <code>"1"^^xsd:integer</code> will not be triggered if the definition uses <code>"1"^^xsd:nonNegativeInteger</code>. The rationale behind exposing this configuration parameter is to allow the users to use their preferred data type.</p>	

Note: The values of some of the initialization parameters directly affect the memory requirements of BigOWLIM repository. The distribution includes an Excel sheet to ease the calculation for the amount of memory that BigOWLIM will use to process the input data ([bigowlim-mem-calc.xls](#)). This file can also be found at: <http://www.ontotext.com/owlim/big/bigowlim-mem-calc.xls>

An example `<repository>` section that may appear in the `system.conf` file of Sesame:

```
<repository id="bigowlim">
  <title>OWLIM with PROTON ontology</title>
  <sailstack>
    <sail class="com.ontotext.tree.big. BigOWLIMSchemaRepository">
      <param name="ruleset" value="owl-horst" />
      <param name="partialRDFS" value="true" />
      <!-- semicolon should be used as delimiter for both parameters -->
      <param name="imports" value=
"/ontology/owl.rdfs;
./ontology/protons.owl;
./ontology/protont.owl;
./ontology/protonu.owl;
./ontology/protonkm.owl;
./ontology/kimso.owl;
./ontology/kimlo.owl"/>
      <param name="defaultNS" value=
"http://www.w3.org/2002/07/owl#;
http://proton.semanticweb.org/2005/04/protons#;
http://proton.semanticweb.org/2005/04/protont#;
http://proton.semanticweb.org/2005/04/protonu#;
```

D2.6.3 / A scalable repository for massive semantic annotation

```

http://proton.semanticweb.org/2005/04/protonkm#;
http://www.ontotext.com/kim/2005/04/kimso#;
http://www.ontotext.com/kim/2005/04/kimlo# " />
  <param name="storage-folder" value="/tmp/big-owlim" />
  <param name="page-cache" value="1000" />
  <param name="entity-index-size" value="4000000" />
  <param name="triple-cache-size" value="4000000" />
  <param name="key-index-size" value="1000000" />
</sail>
</sailstack>
<!--Access Control List can contain zero or more 'user' elements-->
<acl worldReadable="false" worldWritable="false">
  <user login="admin" readAccess="true" writeAccess="true"/> </acl>
</repository>

```

5.3 Sample Memory Configurations

Here follows a table with three configurations, suitable respectively for loading the LUBM benchmark (see section 6) for 50, 5000, and 8000 universities. The table presents the size of the dataset, the cache parameters, and an estimate of the total amount of Java heap memory required.

Table 3 Sample BigOWLIM Cache Configurations and Memory Usage

Parameter	Factor	CFG1, LUBM(50,0)		CFG3, LUBM(5000,0)		CFG2, LUBM(8000,0)	
		Value	Memory Size (mb)	Value	Memory Size (mb)	Value	Memory Size (mb)
key.index.size	16	500 000	8	10 000 000	153	25 000 000	381
cache-size	29	500 000	14	50 000 000	1 383	60 000 000	1 659
entity-index-size	4	1 000 000	4	50 000 000	191	120 000 000	458
page-cache	16 000	1 000	15	10 000	153	10 000	153
<i>total cache</i>			41		1 879		2 651
entity dictionary	29	1 384 243	38	138 424 276	3 828	221 478 842	6 125
literals	29	461 414	13	46 141 425	1 276	73 826 281	2 042
<i>dict+literals</i>		1 845 657	51	184 565 701	5 104	295 305 122	8 167
<i>tree index ref</i>	32	2 000 000	61	2 000 000	61	2 000 000	61
Total heap memory (MB)		153		7 044		10 879	
Explicit RDF statements		6 619 939		661 992 624		1 059 433 190	

6 Performance Evaluation

In order to evaluate the scalability and performance of BigOWLIM, a number of tests and evaluations have been performed. Those based on the LUBM benchmark are presented below; they prove that BigOWLIM scales up to billions of statements on server hardware. Based on the limited publicly available evaluation results, the results indicate that BigOWLIM is the most scalable OWL repository currently available!

Below we present the test hardware configurations (Table 4) and the results of BigOWLIM at different sizes of the LUBM benchmark, [8]. One can find the results of SwiftOWLIM at this benchmark in section 8.3 of [18].

Table 4 Hardware and Software Configurations for the Different Runs

Name	Configuration	RAM (jvm -Xmx)	JDK	Comment
4cOpt12g	2xOpteron 270 (2.0GHz), dual-core Suse Linux v.10, 64-bit	12GB, DDR400	JDK 1.5 64-bit	A database/application server; 4 SATA2 drives on RAID10; assembly cost ~4000 EURO
2Opt6.0g	2xOpteron 246 (2.0GHz) Windows Server 2003 64-bit	6GB, DDR400	JDK 1.5 64-bit	A database/application server; 4 SATA2 drives on RAID10; assembly cost ~3000 EURO
Pdc1.6g	Pentium D 920 (2.8GHz), Win XP	1.6GB, DDR400	JDK 1.5	Workstation
Piv0.9g	Pentium IV 630 (3.0GHz), Win XP	900MB, DDR2 533	JDK 1.5	Office desktop
Pm0.7g	Pentium Mobile 1.6GHz, Win XP	680MB, DDR266	JDK 1.5	Notebook (Q2'03)

6.1 LUBM Benchmark

The Lehigh University evaluation, [8], is one of the most comprehensive benchmark experiments recently published. It evaluates the upload and query performance of four systems: memory-based Sesame, database-based Sesame, DLDB-OWL, and OWLJessKB. The benchmark was made with a relatively simple ontology of the university organizational structure and with synthetically generated datasets – for each university, a number of departments and employees with their descriptions and the relations holding between them are generated. The performance of the various systems is measured in the course of an incremental increase in the number of the universities (from 1 to 50).

LUBM benchmark consist of sample ontologies (and data), Java benchmark programs and “adapter” interfaces, which make possible its easy adoption for different repositories. An “adapter” of OWLIM for LUBM is provided together with the distributions of both SwiftOWLIM and BigOWLIM and documented in [19].

6.2 LUBM(50,0) – about 7 million explicit statements

LUBM(50,0) is the largest dataset which is provided by Lehigh University, so, most third party results are available for this size. It also seems to serve as a “pass” to the club of scalable semantic repositories. This set consists of about 1000 OWL (RDF/XML) files, with a total volume 583MB, including about 6.6 million explicit statements.

We provide first the results of the LUBM evaluation of the default configuration of SwiftOWLIM 2.8.4. Those are meant to serve as a basis for comparison of those of BigOWLIM and can be summarized as follows:

- On a desktop machine (**Piv0.9g**) SwiftOWLIM loads the LUBM(50,0) dataset in 383 sec., i.e. about 6 min.
- All the 14 queries are answered correctly. The query evaluation times for LUBM (50,0) are given in Table 5, together with the size of the results sets.
- The results are correct with respect to the LUBM specification, which means that the default inference setup of OWLIM (rule set **owl-horst** with **partRDFS** optimization) handles all the semantics required for this benchmark.

Table 5 LUBM(50,0) Query Evaluation Time for SwiftOWLIM

Query	Time (ms)	Res.#	Query	Time (ms)	Res.#	Query	Time (ms)	Res.#
1	751	4	6	1 326	519 842	11	1	224
2	1 728	130	7	1	67	12	98	15
3	1	6	8	1 487	7 790	13	1	228
4	1	34	9	4 245	13 639	14	551	393 730
5	9	719	10	1	4			

Recent experiments with SwiftOWLIM 2.9, [20], demonstrate that the new multi-threaded version of the SwiftTREE engine, allows it to load LUBM(50) on the same desktop machine (**Piv0.9g**) within 277 sec. (less than 5 min.). Given a server machine (**4cOpt12g**), SwiftOWLIM can load this dataset for the unmatched 112 sec., i.e. less than 2 minutes.

An experiment was also conducted to detect the largest LUBM dataset which can be handled by SwiftOWLIM on machine **2Opt6.0g** running 64-bit JDK: it was able to load LUBM(300,0) in about 50 min (2943 sec.); the overall size of this dataset is about 40 million explicit statements, which correspond to more than 3GBytes of input files (RDF/XML) and above 6GBytes stored in the N-Triples files used for persistency.

For the run on LUBM(50,0) on desktop machine (**Piv0.9g**) BigOWLIM as configured as in CFG1 (see section 5.3) and given 256MB of Java heap memory. The results are as follows:

- Loading and parsing data files + inference: 1317 sec. (about 22 min.);
- Pure parse time: about 110 sec. (1000 OWL/RDF files, ~0.5Gb);
- Overall Storage files at the end: 679 MB.

The result means that BigOWLIM is approximately 3 times slower than the in-memory version. Anecdotically, BigOWLIM needed about 3 times less memory, as well – to pass LUBM(50,0) with its default configuration SwiftOWLIM requires about 760MB of Java heap.

Some other experiments made with differently Java heap allowances show that LUBM(50,0) with BigOWLIM takes 2160 seconds in 192MB and even less 1055 seconds in 350MB. Thus we conclude that SwiftOWLIM is more appropriate to be

used when the required memory amount is available. However BigOWLIM performs better by means of less memory usage and answers the queries even when the allocated memory is not enough for running SwiftOWLIM. So BigOWLIM is the appropriate solution for applications with limited memory usage but it also runs increasingly faster according to the allowed memory.

6.3 LUBM(1000,0) – above 130 million statements

The LUBM(1000,0) test set was generated and processed on an average desktop hardware, namely **PiV0.9g**. The final figures for the benchmark are as follows:

- Loading and parsing data files + inference: 40,723 sec. (about 11 h. 20 min.);
- Pure parse time: about 15,000 sec. (20 000 OWL/RDF files, ~11,5Gb);
- Overall Storage files at the end: ~12.2 GB.

The query evaluation times are presented in Table 6:

Table 6 LUBM(1000,0) Query Evaluation Time for BigOWLIM

Query	Time (ms)	Res.#	Query	Time (ms)	Res.#	Query	Time (ms)	Res.#
1	1 828	4	6	276 689	10 447 381	11	94	224
2	121 298	2 528	7	110	67	12	125	15
3	109	6	8	375	7 790	13	86 266	4 760
4	281	34	9	361 252	272 982	14	131 689	7 924 765
5	62	719	10	47	4			

6.4 LUBM(5000,0) – above 670 million statements

We also generated the 5000 university dataset and proceeded with the benchmark, but this time on **4cOpt.12g** – a more powerful machine, suitable for database server tasks. And the final figures for this benchmark are:

- Loading and parsing data files + inference: 150 129 sec. (about 41 h. 45 min.);
- Overall Storage files at the end: ~60.0 GB.

The query evaluation times are presented in Table 7:

Table 7 LUBM(5000,0) Query Evaluation Time for BigOWLIM

Query	Time (ms)	Res.#	Query	Time (ms)	Res.#	Query	Time (ms)	Res.#
1	1803	4	6	565 240	52 213 184	11	515	224
2	194 593	2 528	7	21 933	67	12	1 053	15
3	5 497	6	8	10 767	7 790	13	160 891	2 3269
4	30 011	34	9	1 332 268	1 360 788	14	444 123	39 616 787
5	7 775	719	10	7 175	4			

6.5 LUBM(8000,0) – above 1 billion statements

On the same hardware, we also generated the 8000 university corpus just get over a 1 billion explicit triples. The figures about this experiment are:

D2.6.3 / A scalable repository for massive semantic annotation

- Loading and parsing data files + inference: 251 413 sec (about 69 h. 51 min.);
- Overall Storage files at the end: 95.3 GB.

The query evaluation times are presented in Table 8:

Table 8 LUBM(8000,0) Query Evaluation Time for BigOWLIM

Query	Time (ms)	Res.#	Query	Time (ms)	Res.#	Query	Time (ms)	Res.#
1	1 829	4	6	3 033 415	83 557 706	11	1 186	224
2	966 495	2 528	7	65 401	67	12	2 490	15
3	15 688	6	8	25 379	7 790	13	321 145	37 118
4	77 685	34	9	4 736 695	2 178 420	14	2 091 983	63 400 587
5	20 663	719	10	26 212	4			

7 Performance Comparison

Apart from LUBM (presented in section 6) there are few other known benchmarks used for evaluation of semantic repositories – it is our policy that whenever such benchmark becomes available, we test OWLIM on it and report the results, in order to allow for an, as easy and direct, as possible, comparison to other engines. We have also tried to collect any publicly available results regarding the performance of scalable semantic repositories, even in cases when not all the relevant information is available or the repeatability of the experiments was not obvious. The goal was to collect all results, which suggest state-of-the-art performance and scalability.

We compare only the load time of the tools, which in most of the tests includes parsing (of RDF/XML or other formats), storage and indexing, and inference (for the engines, which perform some form of forward-chaining). Although measuring the query evaluation time is also interesting, comparing and analyzing it across different tasks, test setups and systems is a task of preventing complexity. Still, query results are commented whenever those present and indication for the completeness and soundness of the inference implemented by the systems.

The results are given in two big groups according to the number of the processed statements: below and above 10M (million) statements. To our observations, this borderline clearly distinguishes two classes of systems:

- **Reasoners**, which put emphasize on comprehensive reasoning and
- **Repositories**, which are mostly concerned with indexing large volumes of RDF triples.

It is our understanding, that this borderline is also interesting as a sort of entry level for an enterprise class database management system.

We present the comparison information into two tables: Table 9 for the reasoners and Table 10 for the repositories. Each of the rows in these tables includes the name of the task (the benchmark used) and reference to the reported results; the name and version of the tool; the hardware configuration used for testing. Then we present the number of statements and time for loading them; based on this data we compute the system speed in terms of “explicit statements loaded per second”. We also provide a short reference to the allowed/used inference type and complexity. The rightmost column provides comments on the test setup, the system or the completeness of the “run”.

7.1 Up to 10 Million Statements (Reasoners)

As one can see on Table 9, SwiftOWLIM is the fastest machine in this class, while there is no other engine which can deliver performance above 20 000 st./sec., it passes LUBM(50)⁴ at 25 000 st./sec. on a desktop machine (row #10). It is almost twice faster than the second fastest engine AlegraGraph, loading the same dataset (row #9, Table 9). The differences in the setups and functionality seem to be in balance: (i) AG is evaluated on a server machine; (ii) AG supports a richer RDF model; (iii) the reasoning performed by AG is much simpler, which is confirmed by the incomplete query results. To put its position as the fastest OWL engine outside of any doubt, SwiftOWLIM demonstrates a speed above 60 000 st./sec. on server hardware.

⁴ We use LUBM(N) instead of the LUBM(N, 0) to make the presentation more compact.

If memory constraints are considered, one should take into account the LUBM(50) result of BigOWLIM, given 256MB of Java heap on a desktop machine (row #6 in Table 9). BigOWLIM is about 35 times faster than DLDB-OWL [8], running on a machine, which can be estimated to be at most 3 times slower than the desktop BigOWLIM is evaluated on. It also outperforms Sesame (row #4) and Jena (row #5), according to the results achieved in [12] on a better hardware configuration with several times higher memory allowance. AllegroGraph is the only system that performs better than BigOWLIM, but it supports much simpler inference.

The next used benchmark is OUBM Lite-10, [15], which requires more complex inference chain than LUMB – this explains that much lower load speeds can be achieved on a smaller dataset (about 2M statements). Here SwiftOWLIM (row #14) is compared to the results achieved by Minerva (row #12) and DLDB-OWL (row #13) on comparable desktop machines and appeared to be respectively 20 and 50 times faster.

Exquant, [26], 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 specific length (in the results reported here 1 000 or 10 000) of individuals linked through P, then asserts that the last of 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 OWLIM uses forward-chaining and materialization it performs much worse than RacerPro and KAON2 (respectively rows #17, #15, and #16), whose inference strategies “postpone” most of the inference, necessary for this test, for the retrieval (query evaluation) phase. This becomes obvious as, according to [26], KAON2 was not able to answer within 10 minutes even the query for EXQUANT-500. This means that it would take KAON2 much more than 600 sec. to perform the inference necessary for answering the EXQUANT-1000. In comparison, SwiftOWLIM needs 487 sec. to load the example, but only few milliseconds to answer the query. Data for the time necessary for RacerPro to answer the query is not provided in [26].

Considering that examples of such long “transitive chains” can also occur in real datasets, in SwiftOWLIM 2.9 we have developed a hybrid reasoning mode, where the implicit closures of transitive properties are not materialized. Running in this mode (row #18) OWLIM gets 40 times faster; while it is still slower than RacerPro and KAON2 the results are already comparable. The best news about this hybrid reasoning mode is that using it OWLIM scales much better, while the speed decreases linearly to the length of the chain (see row #19).

Another system that should be mentioned here is Triple20, [27]. Tested to load the OWL representation of WordNet (0.5M statements, row #20), it is the only other engine (apart from OWLIM) which can load and perform comprehensive inference at speeds above 5 000 statements/sec.

Table 9 Load Performance Comparison, Up to 10M statements (Reasoners)

#	Task and Test ref.	Tool, Version	Hardware & conf. (CPU, RAM, -Xmx)	Mill. stat.	Time (sec)	Speed st./sec	Inference	Comment
1	LUBM(20), [8]	Sesame-DB v.1.01	P4 1.8Mhz, 256MB, Xmx512, JDK 1.4	2,8	167 753	17	RDFS	Q6-13 incompl.
2	LUBM(20), [8]	DLDB-OWL v.04-03-30	P4 1.8Mhz, 256MB, Xmx512, JDK 1.4	2,8	15 773	176	OWL-Horst-	Q11-13 incompl.
3	LUBM(50), [8]	DLDB-OWL v.04-03-29	P4 1.8Mhz, 256MB, Xmx512, JDK 1.4	6,9	45 477	152	OWL-Horst-	Q11-13 incompl.
4	LUBM(50), [12]	Sesame-Mem v.1.1	2xXeon 3Ghz, 4GB, Xmx1800, JDK 1.4	6,9	2 820	2 443	not tested	
5	LUBM(50), [12]	Jena v2.1	2xXeon 3Ghz, 4GB, Xmx1800, JDK 1.4	6,9	1 571	4 386	not tested	
6	LUBM(50), [here]	BigOWLIM v0.92b	P4 3Ghz, 2GB RAM, Xmx256, JDK 1.5; owl-horst	6,9	1 317	5 232	OWL-Horst +/-	
7	LUBM(50)	BigOWLIM v0.92b	P4 3Ghz, 2GB, Xmx350; owl-horst	6,9	1 055	6 531	OWL-Horst +/-	
8	LUBM(50), [12]	BRAHMS	2xXeon 3Ghz, 4GB, Xmx1800, JDK 1.4	6,9	1 256	5 486	not tested	Loading from image in 3 s.
9	LUBM(50), [6]	AllegroGraph RDFS++ Reasoner 1.2	1xOpt844, 1.8Ghz, 16GB, Xmx2000, JDK 1.5_64bit	6,9	503	13 698	RDFS +/-	Incompl. q2, q5, q7; un-sound q4, q6
10	LUBM(50), [20]	SwiftOWLIM v.2.9b4	P4 3Ghz, 2GB, Xmx900; owl-horst	6,9	277	24 874	OWL-Horst +/-	
11	LUBM(50), [20]	SwiftOWLIM v.2.9 b4	2xOpt270, 2Ghz, 12GB, Xmx2000, JDK 1.5_64bit	6,9	112	61 518	OWL-Horst +/-	
12	OUBM Lite-10, [15]	Minerva v.20051118	P4 2.66GHz, 1GB, Xmx512, JDK 1.4.2	2,2	9 600	232	OWL Lite +/-, forward-chaining	External DL reasoner used to infer class subs.
13	OUBM Lite-10, [15]	DLDB-OWL	P4 2.66GHz, 1GB, Xmx512, JDK 1.4.2	2,2	4 000	550	OWL Lite +/-,back..chaining	External DL reasoner used
14	OUBM Lite-10, [20]	SwiftOWLIM v2.9b4	P4 3Ghz, 1GB, Xmx512, JDK 1.5; owl-max	2,2	192	11 458	OWL Lite -	
15	Exquant 1000, [25]	RacerPro v1.9.0	P4 3Ghz, 1GB, Xmx700, JDK 1.5; through JRacer	0,001	3	333	OWL Lite, through backward chaining	Query answering time not reported
16	Exquant 1000, [25]	KAON2 build 5/12/05	P4 3Ghz, 2GB, Xmx700, JDK 1.5;	0,001	3	333	OWL Lite, through backward chaining	Unable to answer the query within 10 min.
17	Exquant 1000, [20]	SwiftOWLIM v.2.9b4	P4 3Ghz, 2GB, Xmx700; owl-max, non-transitive, 1 thread	0,001	487	2	OWL Lite-, through forward chaining	the result from [25] about v2.8.3 were not repeatable
18	Exquant 1000, [20]	SwiftOWLIM v.2.9b4	P4 3Ghz, 2GB, Xmx700; owl-max, transitive, 1 thread	0,001	11	91	OWL Lite-	Through hybrid inf. Strategy for trans. prop.
19	Exquant 10000, [20]	SwiftOWLIM v.2.9b4	P4 3Ghz, 2GB, Xmx700; owl-max, transitive, 1 thread	0,010	1 076	9	OWL Lite-	Through hybrid inf. strategy for trans. prop.
20	WordNet [27]	Triple20	2xAthlon 1600+, 2GB	0,5	82	5 789	OWL Horst +/-	Claimed 32-bit scalability to 40M st.

7.2 Above 10 Million Statements (Repositories)

There are very few engines which are capable of handling hundreds of millions of RDF statements with inference. W3C represents an open forum where the best achievements in the area are reported and commented – it is the Wiki page.

While comparing the systems in the class of reasoners (see section 7.1) we are interested mainly in the performance speed, since the inference complexity and supported semantics of few of the systems is similar enough. In contrast, in the higher scalability class of systems that are able to process more than 10M statements, the inference complexity appears to be the major differentiating factor. There are several systems which can process hundreds of millions of statements with speed of several thousand of st./sec.

The results reported for the different systems are given in Table 10 ordered by the number of processed statements. The following conclusions can be drawn:

- BigOWLIM is the only machine which can perform inference with OWL-Horst with more than 300M statements.
 - The only machine which performs some inference at this scale is AlegroGraph; unfortunately, the LUBM(50) results published in [6], show that it is neither sound nor complete even with respect to the relatively simple semantics requires by LUBM;
 - Among all the test results summarized in Table 10, RDF Gateway and OWLIM are the only systems, which reason with respect to a logical fragment more complex than OWL-Horst. While AlegroGraph and ORACLE are trying to go above RDF(S), there are no proofs they can get to the level necessary even for LUBM. RDF Gateway is slower and there are no data suggesting that it can match the scalability of BigOWLIM;
- BigOWLIM is the only machine which can handle more than 100M statements on desktop hardware (130M statements, row #5);
- SwiftOWLIM is the fastest engine which can handle more than 100M statements. It outperforms AlegroGraph, although is performs more complicated inference.

Thus we can conclude that based on any published results, BigOWLIM is the best RDF engine with OWL inference support. It clearly outperforms all the competitors in this class in terms of both speed and scalability. BigOWLIM is also one of the very few engines which can handle 1 billion triples.

Contrary to the expectation that SwiftOWLIM is not scalable, there are just a few engines which can match its scale, but no one of them can match its reasoning capabilities. SwiftOWLIM is simply the fastest engine in this evaluation.

Table 10 Load Performance Comparison, Above to 10M statements (Repositories)

#	Task and Test ref.	Tool, Version	Hardware & config. (CPU, RAM, Java Heap, JDK)	Mill. of stat.	Time (sec)	Speed (stat./sec.)	Inference	Comments
1	Synthetic RDF, [28]	KOWARI	P4 2.8GHz, JDK 1.4.2	50	61 200	817	no inf. at load time	Loading single RDF/XML file
2	LUBM(500), [29]	Sesame's Native Store (v. 2.0alpha3)	P4 2.8GHz, 1GB, Xmx800, JDK 1.5	70	10 800	6 481	RDFS +/-	
3	LUBM(600)	SwiftOWLIM v0.92b	2xOpt270, 2Ghz, 16GB, (4cOpt2g), Xmx12000, JDK1.5_64bit; owl-horst	83	3 941	20 979	OWL-Horst +/-	
4	Subset of UniProt, [22]	ORACLE , 10gR2	P4 3.0GHz, 2GB	100	277 200	361	RDFS +	Quadruple model used
5	LUBM(1000)	BigOWLIM v0.92b	P4 3Ghz, 2GB, Xmx1700, JDK 1.5; owl-horst	130	40 723	3 192	OWL-Horst +/-	
6	FOAF,DC, PRISM data, [23]	Jena v2.3	Xeon 3.2GHz, 4GB, 6 SCSI (15k drives in RAID5)	200	no info.		RDFS +/-	Load speed reported to be a problem
7	Synthetic RDF, [28]	KOWARI	AMD Opteron, 1GHz (?), 64-bit JDK; no other info.	235	61 200	3 840	no inf. at load time	[29] claims 500M without details
8	Uniprot, [11]	RDF Gateway	no information	262	345 600	758	OWL Lite, through backward chaining	"the load took several days" approx. as 4 days
9	LUBM(5000)	BigOWLIM v0.92b	2xOpt270, 2Ghz, 16GB, (4cOpt2g), Xmx8000, JDK1.5_64bit; owl-horst	689	150 129	4 589	OWL-Horst +/-	
10	Reif. LUBM + Movie & Actor DB, [6]	AllegroGraph RDFS++ Reasoner 1.2	1xOpt844, 1.8Ghz, 16GB, Xmx2000, JDK 1.5_64bit	1 000	50 580	19 771	RDFS +/-;	
11	LUBM(8000), [21]	OpenLink Virtuoso	2xXeon 5130, 2GHz, 8GB, RAID 4xSATA	1 060	85 560	12 389	none	
12	LUBM(8000)	BigOWLIM v0.92b	2xOpt270, 2Ghz, 16GB, (4cOpt2g), Xmx12000, JDK1.5_64bit; owl-horst	1 060	251 413	4 216	OWL-Horst +/-, complete	

Bibliography and 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] Broekstra, J. (2005). *Storage, Querying and Inferencing for Semantic Web Languages*. Ph.D. Thesis, Vrije Universiteit Amsterdam, SIKS Dissertation Series No. 2005-09, ISBN 90 9019 2360. <http://www.cs.vu.nl/~jbroeks/#pub>
- [4] 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/>
- [5] 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, Feb. 2004. <http://www.w3.org/TR/owl-ref/>
- [6] Franz Inc. (2007). *AllegroGraph 64-bit RDFStore™*. As of Feb 4th, 2007. <http://www.franz.com/products/allegrograph/>
- [7] Groszof, B; Horrocks, I; Volz, R; Decker, St. (2003). *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WWW2003, Budapest, May 2003.
- [8] 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>
- [9] Hayes, P. (2004). *RDF Semantics*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
- [10] Horrocks, I., Patel-Schneider, P. F., Bechhofer, S., Tsarkov, D. *OWL Rules: A Proposal and Prototype Implementation*. [Journal of Web Semantics 3 \(2005\), pp. 23-40.](http://www.websemanticsjournal.org/ps/pub/2005-16)
- [11] IntelliDimension. (2005). *Experiments with Uniprot RDF and RDF Gateway*. Geoff Chappell, 2005-06-16. <http://labs.intelldimension.com/uniprot/>
- [12] Janik, M., Kochut, K. *BRAHMS: A WorkBench RDF Store and High Performance Memory System for Semantic Association Discovery*. In Proc. of ISWC 2005, Galway, Ireland, November 6-10, 2005. LNCS 3729, pp. 431-445.
- [13] 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.
- [14] 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/>
- [15] 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).
- [16] Manov, D; Popov, B. (2006). *Massive Automatic Annotation*. Deliverable D2.6.1 of SEKT project. Jan 2006, <http://www.sekt-project.com/>.
- [17] Motik, B., Sattler, U., Studer, R. *Query Answering for OWL-DL with Rules*. [Journal of Web Semantics 3 \(2005\), pp. 41-60.](http://www.websemanticsjournal.org/ps/pub/2005-16)

D2.6.3 / A scalable repository for massive semantic annotation

- [18] Ontotext Lab. (2006). *OWLIM System Documentation*. Version 2.8.4.
<http://www.ontotext.com/owlim/v2.8.4/OWLIMSysDoc.pdf>
- [19] Ontotext Lab. (2006). *OWLIM Tests and Benchmarks*. Version 2.8.4.
<http://www.ontotext.com/owlim/v2.8.4/OWLIMTests.pdf>
- [20] Ontotext Lab. (2007). *SwiftOWLIM System Documentation*. Version 2.9b3.
<http://www.ontotext.com/owlim/v2.9/OWLIMSysDoc.pdf>
- [21] OpenLink Software. (2006). *Advances in Virtuoso RDF Triple Storage (Bitmap Indexing)*. A wiki publication by Orri Erling (Program Manager - OpenLink Virtuoso), October 31, 2006,
<http://virtuoso.openlinksw.com/wiki/main/Main/VOSBitmapIndexing>
- [22] ORACLE. (2006). "Re: query regarding oracle10g RDF store". ORACLE's Forum reply by Susie Stephens on 20 March 2006. <http://forums.oracle.com/forums/thread.jspa?threadID=371471>
- [23] Parvatikar, P; Portwin, K. (2006). *Scaling Jena in a commercial environment: The Ingenta MetaStore Project*. Jena Users Conference 2006.
<http://jena.hpl.hp.com/juc2006/proceedings/portwin/paper.pdf>
- [24] ter Horst, H. J. *Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity*. In Proc. of ISWC 2005, Galway, Ireland, November 6-10, 2005. LNCS 3729, pp. 668-684.
- [25] W3C. (2007). *LargeTripleStores W3C wiki page*. As of Feb 4th, 2007.
<http://esw.w3.org/topic/LargeTripleStores>
- [26] Weithöner, T., Liebig, T., Luther, M., Böhm, S. (2006). *What's Wrong with OWL Benchmarks?* SSWS2006.
- [27] Wielemaker, J., Schreiber, G., Wielinga, B. (2005). *Using triples for implementation: the {Triple20} ontology-manipulation tool*. ISWC 2005, Galway, Ireland. Springer-Verlag, LNCS 3729.
- [28] Wood, D; Gearon, P; Adams, T. (2005). *Kowari: A Platform for Semantic Web Storage and Analysis*. XTech 2005, May 2005, Amsterdam.
<http://www.idealliance.org/proceedings/xtech05/papers/04-02-04/>
- [29] [Wood, D. \(2005\). *Scaling the KOWARI Metastore*](#). In Proc. of Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA. Springer Verlag, LNCS 3807.
- [30] Yankova, M., Popov, B; Kitchukov, I.; Angelov, K.: SEKT D2.6.2 Massive Automatic Annotation –V2.