OWL Semantic Repository

ver. 2.8.2, 3 Mar 2005

System Documentation

1 Fact Sheet 2

2 Introduction 4

2.1 OWL Layering Problems 4

2.2 OWL Horst: Proper Rule Extension of RDFS 5

2.3 Reasoning Strategies 6

3 TRREE Engine 8

4 OWLIM: a Semantic Repository as Sesame SAIL using TRREE 9

5 Persistence Strategy 10

5.1 Read-only statements 10

6 Supported Semantics 12

6.1 PROTON Primitives Involved 13

6.2 Partial "Optimized" RDFS 13

7 Interfaces and RMI Access 15

8 Installation and Configuration 16

8.1 Distribution Contents 16

8.2 Running Standalone OWLIM for Remote Access 17

8.3 Configuration 17

9 Performance 22

9.1 City Benchmark 22

9.2 LUBM Benchmark 24

9.3 OWLIM Performance Analysis 25

10 References 28

© Copyright 2005, Ontotext Lab, Sirma Group Corp.

1 Fact Sheet

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, OWL DLP, and OWL Horst reasoning, based on forward-chaining of entailment rules. The most expressive language supported is a combination of limited OWL Lite and unconstrained RDFS.

Reasoning and query evaluation are performed in-memory while at the same time, a reliable persistence strategy assures data preservation, consistency, and integrity. OWLIM can manage millions of explicit statements on desktop hardware. A principle limitation of OWLIM is the relatively slow delete operation. The upload, reasoning, and the query evaluation proceed extremely fast even against huge ontologies and knowledge bases. According to the limited evaluation data available, OWLIM is the fastest OWL Lite repository available.

The development of OWLIM is partly supported by project SEKT, EU IST IP 2003-506826.

Availability and Contacts

Version: 2.8.2, 3 Mar 2006.

Download: http://www.ontotext.com/owlim/v2.8.2/owlim-2.8.2.zip

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

The Current Version Release Notes

- TRREE: OWLIM uses TRREE engine for in-memory reasoning and query evaluation. TRREE is a newer version of the IRRE engine, which was part of OWLIM v.2.8.
- 7 different inference modes: OWLIM can be configured to work with one of three predefined sets of rules that support respectively the semantics of RDFS, OWL Horst, and a specific fragment we name owl-max (combining OWL Lite with unrestricted RDFS). These rule sets can be altered to "partial-rdfs" mode, in which some of the normative RDFS entailments are discharged for performance reasons. In addition, the entailment is made optional, so that it is possible to switch it off completely and to use OWLIM as a plain RDF store.
- Extended OWL support: `owl:oneOf`, `owl:minCardinality`, `owl:maxCardinality`, `owl:cardinality`; partial OWL-Lite T-Box (schema-level) reasoning added.
- Configurable index size: allows the user to manage the tradeoff between required RAM and performance.

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:

OWLIM is bound to the data and query standards supported by Sesame. RDF is the basic data standard, [12]; the supported query languages are: SeRQL, RQL, RDQL.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

Syntaxes: OWLIM uses N-Triples RDF syntax for persistence. The import and export of all major RDF syntaxes (XML, N3, N-Triples) is supported by Sesame.

Semantics: OWLIM 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 **`RDFSchemaRepository`** interface. OWLIM v2.8.2 was tested with Sesame releases 1.2.1-1.2.4.

TRREE (http://www.ontotext.com/trree/) is a Triple Reasoning and Rule Entailment Engine, which allows forward-chaining and materialization with respect to entailment rules. Within OWLIM, TRREE v2.8.2 comes preconfigured with four distinct sets of rules.

Licensing

OWLIM License Agreement:

(c) Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.
135 Tsarigradsko Shosse, Sofia 1784, Bulgaria, http://www.ontotext.com.

This library is a free software. One can redistribute and/or modify it under the terms of the GNU Lesser General Public License (LGPL) published by the Free Software Foundation. Consult either version 2.1 of the License, or a later. The library is distributed with the hope that it will be useful, but note: WITHOUT ANY WARRANTY, this means- even without the implied warranty of MERCHANTABILITY or of FITNESS FOR A PARTICULAR PURPOSE. For more details, see the GNU Lesser General Public License. It is expected that you received a copy of the GNU Lesser General Public License together with this library; if not, send a request to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Licensing of Third Party Libraries:

Sesame - (c) Copyright Aduna b.v. Sesame is an open-source library, available under LGPL.

TRREE – (c) Copyright, Sirma Group Corp. TRREE is not an open-source library. It is owned by Ontotext Lab. TRREE v2.8.2 is licensed for use free of charge as an integral part of OWLIM v2.8.2. Re-distribution of TRREE in any form, except as part of the original OWLIM distribution package, is strictly forbidden. Any form of modification or reverse-engineering of TRREE is strictly forbidden. TRREE is distributed together with OWLIM without any warranty.

Installation and Usage

OWLIM is distributed as a ZIP archive that contains the **`owlim-2.8.2.jar`**, all the required libraries, configuration files, sources, documentation, and sample data. OWLIM is designed to be used as a Storage and Inference Layer (SAIL) of Sesame – no OWLIM specific interfaces exist. Since Sesame is not included in the OWLIM distribution, it should be downloaded separately.

Future Plans

- A much more scalable disk-based repository (able to manage approx. 200M explicit statements on a desktop hardware);
- SPARQL support;
- Gathering statistics for usage of rules in order to optimize entailment.

2 Introduction

Throughout this document the term *semantic repository* is used to refer to a system for storage, querying, and management of structured data with respect to ontologies. At present there is no single well-established term for such engines. Weak synonyms are: reasoner, ontology server, metastore, semantic/triple/RDF store, database, repository, knowledge base. Usually, the different wording also reflects difference in the implementation, performance, intended application, etc. Introducing the term "semantic repository", we are trying to cover the core functionality offered by most of these tools.

Semantic repositories can be used as a replacement for the database management systems (DBMS), offering easier integration of diverse data and more analytical power. In a nutshell, a semantic repository can dynamically interpret metadata schemata and ontologies, which define the structure and the semantics related to the data and the queries. Compared to the approach taken in the relational DBMS, this allows for (i) easier changes to and combinations of data schemata and (ii) automated interpretation of the data. The latter means that, for example, given a simple ontology, a semantic repository can return a mobile operator in the UK, when queried for telecom companies in Europe.

Over the last decade the Semantic Web[1] emerged as an area where semantic repositories become as important as the HTTP servers are today. This perspective boosted the development, under W3C driven community processes, of a number of robust metadata and ontology standards. Those standards play the role, which SQL had for the development and spread of the relational DBMS. Although designed for Semantic Web, these standards face increasing acceptance in areas like Enterprise Application Integration and life sciences.

The remainder of this section provides introduction in a number of relevant topics, namely the features of the different OWL dialects and reasoning strategies.

2.1 OWL Layering Problems

In order to match the expectations for a useful heap of ontologies and structured metadata, the Semantic Web requires scalable high-performance storage and reasoning infrastructure. The major challenge towards building such an infrastructure is the expressivity of the underlying standards: RDF(S), [12] [4], and OWL, [5]. Eventhough RDF(S) can be considered a simple Knowledge Representation (KR) language, it is already a challenging task to implement a repository for it, one that provides performance and scalability comparable to those of entry-level relational database management systems (RDBMS). Going up the stairs of the Semantic Web specifications stack, the challenges for the repository engineers are getting more and more serious. Even the simplest dialect of OWL (OWL Lite) is a description logic (DL) formalism with no algorithms enabling efficient inference and query answering over reasonably scaled knowledge bases (KB). Furthermore, the semantics of OWL Lite and DL are incompatible with that of RDF(S), see [6]. This causes lack of "backward compatibility". Imagine the situation when an application, which uses RDFS schemata and RDFS-compliant repository, should be "upgraded" to OWL. The evolution should start with the replacement of the RDFS schemata with the OWL ontologies and adoption of a repository supporting (the corresponding part of) OWL.

[1] http://www.w3.org/2001/sw/

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

Even the most direct translation (re-labeling the **rdfs:Class**-es to **owl:Class**) of the schema, without adding further complexity, can lead to different inference and to inconsistencies.

Logical programming (LP) is a common name used for rule-based logical dialects and systems, such as PROLOG, Datalog and Flora 2. Currently OWL DLP is emerging as a new dialect, offering a promising compromise between expressive power, efficient reasoning, and compatibility. It is defined in [6] as the intersection of the expressivity of OWL DL and LP. In fact, OWL DLP is defined as the most expressive sub-language of OWL DL, which can be mapped to Datalog. OWL DLP is simpler than OWL Lite. The alignment of its semantics to the one of RDFS is easier, as compared to the Lite and DL dialects. Still, this can only be achieved through the enforcement of some additional modelling constraints and transformations. A broad collection of information related to OWL DLP can be found at http://logic.aifb.uni-karlsruhe.de/.

2.2 OWL Horst: Proper Rule Extension of RDFS

In [16] 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, [8], 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.

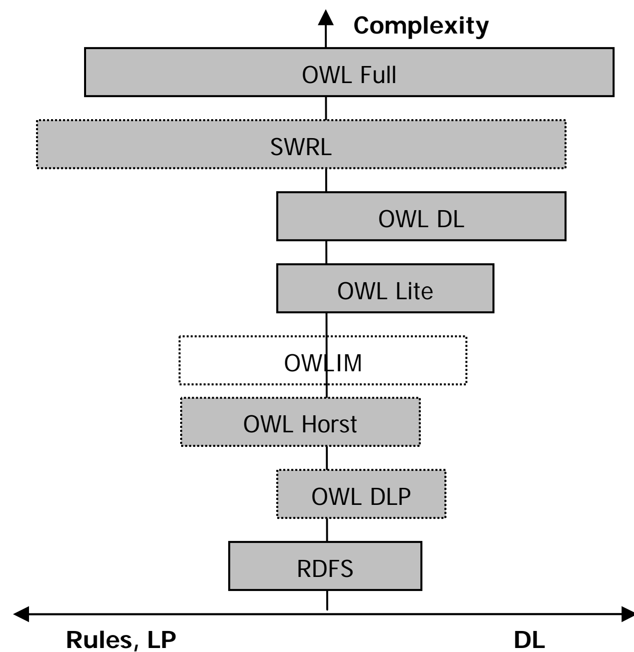

Figure 1. OWL-Related Languages Map

Horst extends and modifies the D-entailment rules from [8] 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-

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

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).

In this document we refer to this extension of RDFS as "OWL Horst". As outlined in [16], 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, [9] and [14], 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 [16].

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 8.3 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 found in section 6.

OWL Horst is close to what has been intuitively described as OWL Tiny at the SWAD-Europe[2] 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 datatypes.

2.3 Reasoning Strategies

The two principle strategies for rule-based inference are, as follows:

- **Forward-chaining**: to start from the known facts (the explicit statements) and to perform inference in an inductive fashion. The goals of such reasoning can vary: to compute the *inferred closure*[3]; to answer a particular query; to infer a particular sort of knowledge (e.g. the class taxonomy).
- **Backward-chaining**: to start from a particular fact or a query and to verify it or get all possible results, using deductive reasoning. In a nutshell, the reasoner decomposes (or transforms) the query (or the fact) into simpler (or alternative) facts, which are available in the KB or can be proven through further recursive transformations.

Both of these strategies have different strong and weak points, which are studied well in the history of KR and expert systems. Hybrid strategies (involving partial forward- and backward-chaining) are also possible and proven to be efficient in many contexts.

---

[2] 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] The term *inferred closure* is defined as follows: the extension of a KB (or a graph of RDF triples) with all the implicit facts (triples), which could be inferred from it, based on the enforced semantics.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

Let us imagine a repository which performs total forward-chaining, i.e. it tries to make sure that, after each update to the KB, the inferred closure is computed and made available for query evaluation or retrieval. This strategy is generally known as *materialization*. In order to avoid ambiguity with various partial materialization approaches, let us call such an inference strategy, taken together with the monotonic entailment[4] assumption, *total materialization*.

The principle advantages and disadvantages of the total materialization are discussed at length in [3]; here we provide just a short summary of them:

- Upload/store/addition of new facts is relatively slow, because the repository is extending the inferred closure after each transaction for modification. In fact, all the reasoning is performed during the upload;
- Deletion of facts is also slow, because the repository should remove from the inferred closure all the facts which are not true any longer.
- The maintenance of the inferred closure usually requires considerable additional space (RAM, disk, or both, depending on the implementation);
- Query and retrieval are fast, because no deduction, satisfiability checking, or other sorts of reasoning are required. The evaluation of the queries becomes computationally comparable to the same task for relation database management systems (RDBMS).

Probably the most important advantage of the inductive systems, based on total materialization, is that they can easily benefit from RDBMS-like query optimization techniques, as long as all the data is available at query time. The latter makes it possible for the query evaluation engine to use statistics and other means in order to make "educated" guesses about the “cost” and the result cardinality for a particular constraint. These optimizations are much more complex in the case of deductive query evaluation.

Total materialization is adapted as a reasoning strategy in a number of popular Semantic Web repositories, including some of the standard configurations of Sesame, http://www.openrdf.org, and Jena, http://www.hpl.hp.com/semweb/jena.htm.

[4] Under a monotonic logic, when new explicit facts (statements) are added to the KB (repository), this can cause that new implicit facts can extend its inferred closure, but in no case facts which were part of the inferred closure before, should be removed. In other words addition of new facts can only monotonically extend the inferred closure.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

3 TRREE Engine

TRREE, http://www.ontotext.com/trree/, stands for Triple Reasoning and Rule Entailment Engine. It is implemented in Java and its previous name was IRRE. TRREE performs reasoning based on forward-chaining of entailment rules over RDF triple patterns with variables. TRREE's reasoning strategy is total materialization, as introduced in the previous section. The rule format and the semantics enforced is analogous to R-entailment (see [16] and the previous section) 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. This leads to lower complexity as compared to R-entailment;
- **`[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 with a set of 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. Reasoning and query evaluation are performed in-memory; the latter means that the full content of the repository is loaded and maintained in a proprietary representation in the main memory, which allows for efficient retrieval.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

4 OWLIM: a Semantic Repository as Sesame SAIL using TRREE

OWLIM[5] is a high-performance semantic repository with support for OWL reasoning and rule extensions. OWLIM uses the TRREE engine (section 3) for forward-chaining of entailment rules. The reasoning framework is analogous to the R-entailment, [16], which is an extension of the D-entailment, defining the standard RDF(S) semantics, [8]. The fragment of OWL supported is discussed in section 6.

OWLIM 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. Since Sesame is one of the most mature and popular semantic repositories, this allows for very easy adoption of OWLIM. OWLIM v.2.8.2 is packaged as a Storage and Inference Layer (SAIL) for Sesame (v.1.2.2-1.2.4) named **`OWLIMSchemaRepository`**; it implements the **`RdfSchemaRepository`** interface. More information related to various aspects of Sesame's specification, architecture, and implementations can be found in [1]. OWLIM is an open-source Java library available under LGPL license.

In OWLIM, reasoning and query evaluation are performed in-memory by the TRREE engine. The reasoning strategy is total materialization (section 2.3). At the same time OWLIM implements a robust persistency and backup strategy, as presented in section 5. The efficiency of TRREE allows OWLIM to manage millions of explicit statements even on desktop hardware. The upload and query evaluation are fast even for huge ontologies and knowledge bases, as demonstrated through a number of benchmarks, presented later in this paper. The major limitation of OWLIM is the relatively slow delete operation.

The easiest way to use OWLIM is in the so-called embedded mode, i.e. as a Java library. The distribution of OWLIM contains a RMI factory (**`CustomRMIFactory`**) that provides RMI access to the SAIL layer of the repository. More details on this are given in section 7. The installation and configuration of OWLIM are discussed in section 8. More information on the various aspects of the Sesame specifications, its architecture and implementations can be found in [3].

---

[5] http://www.ontotext.com/owlim/

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

5 Persistence Strategy

The persistency of OWLIM is implemented by means of N-Triples files. It is based on the same strategy employed in the old version of **RdfSchemaRepositoryV2** SAIL of Sesame that was developed by Ontotext in 2004. The repository can be configured to spread over several files. One of these files, let us name it **persist**, is considered both as an input for the initial loading and as a target for modifications: new statements can be stored in it; old statements can be deleted from it. All other files are considered read-only. In this way, it is possible to implement a scenario with multiple static bodies of background knowledge and data (ontologies, knowledge bases, etc.) and a single file that corresponds to the dynamic content of the repository. Setup of this kind is useful if using layered and modularized ontologies or KB. The configuration of the repository is discussed in section 8.

The implemented backup strategy guarantees that in case of electric power shortage, or any other abnormal termination, no inconsistency or loss of newly asserted triples will occur. The backup strategy works in the following manner:

- All new explicit triples persist in an auxiliary file (specified in the configuration file with the **new-triples-file** parameter) after the write buffer was filled up or after the **commitTransaction()** method was invoked. In the current version of OWLIM, it is not possible to modify the size of this specific write buffer; the default size is 20,000 statements.
- The synchronization of the in-memory contents of the repository and the **persist** file is performed either on a regular basis (in case timeout was configured using the **syncDelay** parameter), or after OWLIM has been properly shutdown.
- In cases when the 'regular' synchronization completes, the **new-triples-file** is converted to a file with a **.~bak** extension and a new empty file is created in its place.
- In case of abnormal process termination, the content of the **new-triples-file** is added to the repository next time the SAIL is initialized, right after the processing of the **persist** file is completed.

A separate thread is used to take care of the persistence to the temporary triple storage, in order to reduce the latency of OWLIM. Delete operations, when invoked in a transaction, only mark all the affected statements, while the actual job is done when the transaction is committed. This strategy allows for combination of reliability with low latency.

Though relatively simple, the described strategy had proven to be very efficient and reliable in the couple of years when **RdfSchemaRepositoryV2** and OWLIM were used as semantic repositories for different applications of KIM (KIM is a platform for semantic annotation, indexing, and retrieval, http://www.ontotext.com/kim).

5.1 Read-only statements

The explicit statements, imported from the "background knowledge" files (that were described in the SAIL configuration using **import** parameter), are treated as 'read-only' and cannot be removed. In this way, the possibility that the applications edit ontologies and schemata, beyond their authority, is blocked. All other explicit statements (the ones imported from the **persist** file or the programmatically added ones) can be manipulated at will.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

For consistency reasons, the axiomatic statements (i.e. the ones inferred from an empty repository, and that had never been formally asserted) are also treated as read-only in accordance with the standard Sesame behavior.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

6 Supported Semantics

Within OWLIM, TRREE is configured by a set of rules and axioms that encompasses the model theoretic semantics of RDFS, as defined in [8], extended with support for the following OWL constructs: **SymmetricProperty**, **TransitiveProperty**, **inverseOf**, **equivalentClass**, **equivalentProperty**, **sameAs**, **FunctionalProperty**, **InverseFunctionalProperty**, **onProperty**, **allValuesFrom**, **someValuesFrom**, **hasValue**, **unionOf**, **intersectionOf**, **differentFrom, oneOf, AllDifferernt, minCardinality, maxCardinality, cardinality**. The major limitations for the support of these primitives are as follows:

- 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:integer**; literals typed as **xsd:nonNegativeInteger** are not properly handled.

Regarding OWL compliance, OWLIM supports a dialect similar to OWL Horst, as defined in [16] and introduced in section 2.2. The following statements can be made for the OWL support in OWLIM (considering the configuration with richest OWL support and no RDF 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 is a proper sub-set of 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 – it is more expressive than OWL Lite in some aspects;
- OWLIM supports a language richer than OWL DLP;

The differences between the OWL dialect supported 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 intuition that the performance "penalty" is too high for most applications;
- There are no inconsistency rules, as those defined in [16] for **disjointWith** and **differentFrom.**
- Few more OWL primitives are supported by OWLIM, namely: **unionOf**, **intersectionOf**, **AllDifferent, oneOf, minCardinality, maxCardinality, cardinality**.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

- 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.

Eventhough the concrete rules, pre-defined in OWLIM, differ from the ones defined in OWL Horst in [16], the complexity and decidability results reported for R-entailment are relevant for TRREE and OWLIM. 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-entailement OWL test cases, [4]. These checks are available as JUnit tests together with the OWLIM distribution; they are documented in [15] 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 8.3) 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 inference. The separation of the rules into the different levels is clearly indicated (through comments) in the file **rules.pie**.

6.1 PROTON Primitives Involved

Besides the RDF- and OWL-specific primitives, the rules, predefined in OWLIM, provide semantics for primitives from the PROTON ontology, http://proton.semanticweb.org. The most important example is the rule that supports **transitiveOver** property, which is defined as follows:

```
Id: proton_TransitiveOver
     p   <protons:transitiveOver>   q
     x   p   y
     y   q   z
    -------------------------------
     x   p   z
```

This property is used later on for alternative definition of the semantics of some RDFS primitives. These definitions support semantics, equivalent to the one given, for instance, with the normative axioms and rules in [8], but allow for better inference speed. They have no "side-effects" on the entailment, unless the corresponding PROTON properties are being used by the application. **transitiveOver** is also used for definition of the semantics of some of the OWL primitives.

The rules supporting the semantics of the PROTON primitives are clearly marked in the **rules.pie** file.

6.2 Partial "Optimized" RDFS

The **partialRDFS** parameter of OWLIM allows for switching on and off an optimization of the RDFS inference. Its purpose is to suspend part of the RDFS entailments, which are useless for many datasets and applications, but require considerable inference resources. In essence, the

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

entailment of statements like **<?X, rdf:type, rdfs:Resource>** is suspended**,** as well as ones inferring membership to **rdfs:Class** and **rdf:Property**.

These optimization is based on the following assumptions:

- **Proper-Schema**: all classes and properties are explicitly defined as such in the RDFS schema or the ontology in OWL. Meaning that "system" properties (such as **rdfs:subPropertyOf**) are only used for such schema elements. In this case, axioms such as **<rdfs:subClassOf, rdfs:domain, rdfs:Class>** are obsolete.
- **Type-Resource-Obsolete**: statements of the sort **<URI, rdf:type, rdfs:Resource>** represent no interest. The same holds for class-membership in **rdfs:Literal** and **rdf:List**. Such statements can be generated for each URI, if necessary.

The Proper-Schema assumption allows for discharging of rule **rdf1** of [8], as well as the following axioms:

```
<rdfs:domain> <rdfs:domain> <rdf:Property>
<rdfs:range> <rdfs:domain> <rdf:Property>
<rdfs:subPropertyOf> <rdfs:domain> <rdf:Property>
<rdfs:subClassOf> <rdfs:domain> <rdfs:Class>
<rdf:type> <rdfs:range> <rdfs:Class>
<rdfs:domain> <rdfs:range> <rdfs:Class>
<rdfs:range> <rdfs:range> <rdfs:Class>
<rdfs:subPropertyOf> <rdfs:range> <rdf:Property>
<rdfs:subClassOf> <rdfs:range> <rdfs:Class>
```

The Type-Resource-Obsolete assumption allows for discharging of rules **rdfs4a** and **rdfs4b** of [8], as well as the following axioms:

```
<rdf:type> <rdfs:domain> <rdfs:Resource>
<rdfs:member> <rdfs:domain> <rdfs:Resource>
<rdf:first> <rdfs:domain> <rdf:List>
<rdf:rest> <rdfs:domain> <rdf:List>
<rdfs:seeAlso> <rdfs:domain> <rdfs:Resource>
<rdfs:isDefinedBy> <rdfs:domain> <rdfs:Resource>
<rdfs:comment> <rdfs:domain> <rdfs:Resource>
<rdfs:label> <rdfs:domain> <rdfs:Resource>
<rdf:value> <rdfs:domain> <rdfs:Resource>
<rdf:subject> <rdfs:range> <rdfs:Resource>
<rdf:predicate> <rdfs:range> <rdfs:Resource>
<rdf:object> <rdfs:range> <rdfs:Resource>
<rdfs:member> <rdfs:range> <rdfs:Resource>
<rdf:first> <rdfs:range> <rdfs:Resource>
<rdf:rest> <rdfs:range> <rdf:List>
<rdfs:seeAlso> <rdfs:range> <rdfs:Resource>
<rdfs:isDefinedBy> <rdfs:range> <rdfs:Resource>
<rdfs:comment> <rdfs:range> <rdfs:Literal>
<rdfs:label> <rdfs:range> <rdfs:Literal>
<rdf:value> <rdfs:range> <rdfs:Resource>
```

For each specific context of usage of OWLIM, one should decide whether the RDF "optimization" assumptions described above are in place and if that is not the case, the optimization should be switched off.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

7 Interfaces and RMI Access

OWLIM can be used with the standard interfaces of Sesame. Documentation of these interfaces (including Javadoc) can be found at http://www.openrdf.org, see [1]. In a nutshell, an application can use Sesame in two ways:

- Higher level interfaces which allow for generic operation and query answering;
- Through the SAIL interfaces, which allow low-level access to a triple repository.

The so-called "Custom RMI factory" is shipped with OWLIM to allow easy and efficient remote access to Sesame from other Java processes. To enable the use of the custom RMI factory, one should specify it explicitly into the **rmi-factory** tag of the **system.conf** file, as in the following example:

```
<rmi-factory enabled="true"
   class="com.ontotext.util.rmi.CustomRMIFactory"port="1099"/>
```

It will act as the default factory and, note, it will return not only the instances that implement **SesameRepostiory**, but also the instances that implement the **com.ontotext.util.rmi.SailAccessor** interface. An illustration of how to access the top SAIL interface of a repository through RMI, when the **CustomRMIFactory** is server-side enabled, is given with the following fragment of code fragment:

```
...
import com.ontotext.util.rmi SailAccessor;
...
SesameService ss = Sesame.getService(
new URI("rmi://localhost:1098"));
ss.login("admin", "REPLACE_ME");
SesameRepository sr = ss.getRepository("kim");
Sail topSail = null;
if (sr instanceof SailAccessor)
topSail = ((SailAccessor)sr).getSail();
```

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

8 Installation and Configuration

OWLIM is a specific plug-in for Sesame. To configure, run, and use OWLIM means to do so for a specific configuration of Sesame. Please refer to the online documentation of Sesame, [1].

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 JAR file (Java library) should be included in the Java class-path of the application. The file can be found in the **`lib`** subfolder of the OWLIM distribution.

NOTE: The OWLIM jar must be placed in the class-path of the application before **`sesame.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));
}

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

The use of OWLIM 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.

8.1 Distribution Contents

The distribution of OWLIM includes:

- **`lib`** folder: contains OWLIM 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;
- **`src`** folder: the Java sources of OWLIM and the RMI factory;

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

- **test** folder: benchmarks and unit tests; contains sources, data and documentation of tests and benchmarks defined in [13]. The two major sample applications are the LUBM benchmark and JUnit tests that implement checks for the normative Positive and Negative-entailment OWL test cases, [4]. They could also be used as sample applications;
- **owlim-control.cmd** file: see section 8.2.
- **rules.pie** file: contains description of the axioms and rules which determine the semantics of OWLIM. The different rule sets and reasoning options are clearly indicated. The file is provided just for information; the compiler necessary to generate TRREE-based reasoners from such files is not part of this distribution.

8.2 Running Standalone OWLIM for Remote Access

In short, before you try to access Sesame from within the application, start the Sesame service and bind it to RMI registry. To enable Sesame to close properly, shutdown the service properly. Even though OWLIM can function properly after an abnormal termination of the process and will not cause any inconsistencies in the repository, we recommend that the proper Sesame shutdown mechanism is used (since it affects the initialization time for the next run).

The distribution of OWLIM contains a sample configuration and a (Windows) **cmd** script, which allows start-up and shutdown of OWLIM. A Sesame distribution should also be obtained and installed beforehand. The concrete steps are as follows:

1. Make sure that Java 1.4.2 or later is installed;
2. Download Sesame v1.2.2-1.2.4 from http://www.openrdf.org;
3. Unpack Sesame at appropriate place and modify the **owlim_control.cmd**, so that the **SESAME_LIB** environment variable points to the folder that contains the jar files from the Sesame distribution;
4. To start OWLIM, type "**owlim_control.cmd start**" and, optionally, provide the pathname of the configuration file and the number of the TCP/IP port to bind it up to. The default configuration file is located at the relative path **./test/test/sesame.conf** in the distribution. The default port is 1098.

OWLIM-configured Sesame will be started in a standalone mode. To shut it down, use the "**owlim_control.cmd stop <portNo>**" shell command, where the port number is also optional.

As background knowledge, the sample configuration file (part of the OWLIM distribution) loads the PROTON ontology in the default repository **owlim** and uses **./kb/kb.nt** file as **persist** file for that repository. The configuration file is provided as a sample; it is expected that the user will modify it, e.g. by providing his/her own ontologies and data as a "background knowledge", if such exists. To start with a clear repository, remove the PROTON specific parts from the configuration file. These are declared in **imports** and **defaultNS** parameters of the **owlim** repository definition. To get familiar with the basics of how to configure Sesame in order to use OWLIM, please, read carefully the next section.

8.3 Configuration

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

Sesame is configured in an XML configuration file, which can be provided also at the stage of retrieving the **SesameService** instance. In case of a remote use, Sesame is to be configured at the server side. 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 OWLIM configuration parameters, with their default and allowed values, are listed in the next table together with a short description of each parameter.

Name	Default	Allowed Values
file	No default value	Any valid file name
	Specifies the NTriples (**.NT**) file, where the repository contents are persisted (see section 3 for explanation of the persistency strategy and the role of this file). Example: `<param name="file" value="./kb/kb.nt" />`	
dataFormat	**ntriples**	**n3, rdfxml, turtle, ntriples**
	Specifies the serialization format for the main persist file. Example: `<param name="dataFormat" value="ntriples" />`	
compressFile	**no**	**yes, no**
	Specifies whether a compression on the **file** will be used. Example: `<param="compressFile" value="no"/>`	
noPersist	**false**	**false**, **true**
ruleset	**owl-horst**	**owl-max, owl-horst, rdfs, empty**
	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: **owl-max** including all the rules and axioms included in the **rules.pie** file. This setting provides support for OWL Lite (with some limitations) and RDFS. See section 6 for a detailed explanation; **owl-horst** provides support for RDFS and an OWL dialect referred here as OWL Horst. See sections 2.2 and 6 for details; **rdfs** including only the RDFS entailment rules equivalent to the ones given in [8]; the rules presented in section 6 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	

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

	OWLIM functions as a plain RDF store without inference; Example: `<param name="ruleset" value="owl-max"/>`	
`partialRDFS`	**`true`**	**`false`**, **`true`**
	This parameter switches on (when set to **`true`**) and off the optimization of the RDFS inference described in section 6.2. 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 **`<?X, rdf:type, rdfs:Resource>,`** as well as such inferring membership to **`rdfs:Class`** and **`rdf:Property`**. This optimization has no effect, when the **`ruleset`** parameter is set to **`empty`**; it has the same effect for all the other rule-sets.	
`indexSize`	4,000,000	Limited by the memory available to the Java virtual machine. Values lower than 100,000 are ignored.
	Controls the initial size of the primary triple index. Its default value is convenient for repositories that hold approx. 5,000,000 explicit statements. One should alter this value in case that the target size differs considerably. A smaller value of **`indexSize`** will reduce the amount of memory used for the index, but such setting will slow down the repository operation as the volume of the data grows up. The amount of memory (in bytes) used for this index can be calculated as **`16*indexSize`**. With the default setting OWLIM allocates 64MB of memory for its primary triple index. Example: `<param name="indexSize" value="4000000"/>`	
`newTriplesFile`	No default value	Any valid file name
	The parameter specifies the name of a file, where OWLIM temporarily stores the new triples recently added to the repository. It is used in case of abnormal termination, so that during the initialization its contents will be automatically added to the repository right after the contents of the main persist file (see parameter **`file`**). In case that OWLIM was properly shut down or successfully synchronized with the main **`persist`** file, the contents of the **`new-triples-file`** becomes superfluous, since each triple, mentioned there, is already included in the main **`persist`** file. If this parameter is omitted in the configuration of SAIL, the backup strategy is set to **`off`** and the repository contents will persist only in case the SAIL was properly shutdown. This would happen if the repository **`shutdown()`** method is invoked. `<param="new-triples-file" value="./kb/new-temp-triples.nt"/>`	

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

baseURL	No default value	Any valid URL
	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. `<param="base-URL" value="http://www.ontotext.com/kim/2004/12/wkb#"/>`	
imports	No default value	Semicolon-delimited list of file names
	A list of schema files which will be imported – all the statements, found in these files, will be loaded in the repository and will be treated as read-only. The serialization format is assumed to be RDFS/XML, unless the file has a **.NT** extension. Example: `<param name="imports" value="owl.rdfs;protons.owl;protont.owl"/>`	
defaultNS	None	Semicolon-delimited list of URLs. The number and order should match this in the **imports** parameter.
	Specifies the default namespaces for each of the imported files (see **imports** parameter). Example: `<param name="defaultNS" value="namespaces list for the imported files" />`	

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

```
<repository id="owlim">
    <title>OWLIM with PROTON ontology</title>
    <sailstack>
        <sail class="org.openrdf.sesame.sailimpl.OWLIMSchemaRepository">
            <param name="file" value="./kb/kb.nt"/>
            <param name="compressFile" value="no"/>
            <param name="dataFormat" value="ntriples" />
            <param name="new-triples-file" value="./kb/new-temp-triples.nt"/>
            <param name="ruleset" value="owl-horst" />
            <param name="partialRDFS" value="true" />
            <param name="indexSize" value="4000000" />
            <!-- 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#;
                http://proton.semanticweb.org/2005/04/protonkm#;
                http://www.ontotext.com/kim/2005/04/kimso#;
```

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

```
              http://www.ontotext.com/kim/2005/04/kimlo#" />
        </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>
```

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

9 Performance

As discussed in section 2.3, several concerns normally arise for reasoners that implement the approach taken in TRREE, and thus in OWLIM. Total materialization, performed in-memory, questions the upload speed for large knowledge bases, as well as the maximal scale achievable (limited by the RAM available) and the speed of the delete operation. In order to evaluate the scalability and performance of OWLIM, a number of tests and evaluations have been performed. Some of them are available as sample applications within the distribution and are documented in [15]. A couple of them are presented below; they prove that OWLIM scales up to millions of statements even on "commodity" desktop hardware. Based on the limited publicly available evaluation results, the results indicate that OWLIM is the fastest OWL repository currently available!

9.1 City Benchmark

To explore the scalability of OWLIM, we conducted a benchmark experiment on several systems with different hardware specifications, as presented in Table 1.

Table 1. Hardware and Software Configurations for the Different Runs

Name	**Configuration**	**RAM** (-xmx)	**JDK**
2Opt6000m	2xOpteron 2.0GHz, Win 2003 64-bit	6000mb, DDR400	JDK 1.5 64-bit
2Opt2600m	2xOpteron 2.4GHz, Red Hat Linux v.3	2600mb, DDR333	JDK 1.5 64-bit
2Xeo1600m	2xXeon 2.4GHz, Win XP	1600mb, DDR333	JDK 1.5 32-bit
Piv900m	Pentium IV, 3.0GHz, Win XP	900mb, DDRII 533	JDK 1.5 32-bit
PM700m	Pentium Mobile 1.6GHz, Win XP	680mb, DDR266	JDK 1.5 32-bit

The test scenario can be summarized as follows:

- initially the repository is populated with about 500k explicit statements – a real ontology (PROTON, http://proton.semanticweb.org) and a knowledge base – the small version of KIM WKB (World Knowledge Base), http://www.ontotext.com/kim;
- the repository is then extended with synthetic descriptions of cities; each transaction is adding the description of a single city, which consist of about 10k explicit statements. The cities are linked to real provinces (randomly chosen from the WKB). Ten synthetic organizations are created and "located" in each city. Each organization is described as active within one industry sector – also randomly chosen from the WKB. Finally, 38 persons are created and specified to occupy different positions at any of the organizations. This way WKB is being iteratively extended with LDAP-like data in a realistic manner.
- A couple of test queries (in SeRQL) are evaluated after the addition of each of the 10 cities (i.e. after roughly 100,000 statements). The queries are discussed below.
- Delete was also tested after each 100 generated cities. The test runs until a certain number of cities or memory limit is reached.

The first query (Q1) is relatively simple – a join of 11 triple patterns. It returns a fixed small set of results, which allows for observation of the pure query evaluation time disregarding the fetch

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

time, which could be significant (for large results) and confusing (when the size varies). The second query (Q2) is more complex - it is composed of pattern of 12 statements and a LIKE "*xyz*" literal constraint. The result set of Q2 grows in linear dependency to the size of the repository and reaches tens of thousands of statements, which makes it a good combined measure for query evaluation and result retrieval time.

The results can be summarized as follows:

- The upload speed (including inference and storage) varies within the range of 20,000-120,000 statements per second, depending on the machine and the size of the repository – see Figure 2. Machine 2Opt6000m shows lower speeds above 25 million statements, only due to lack of RAM;

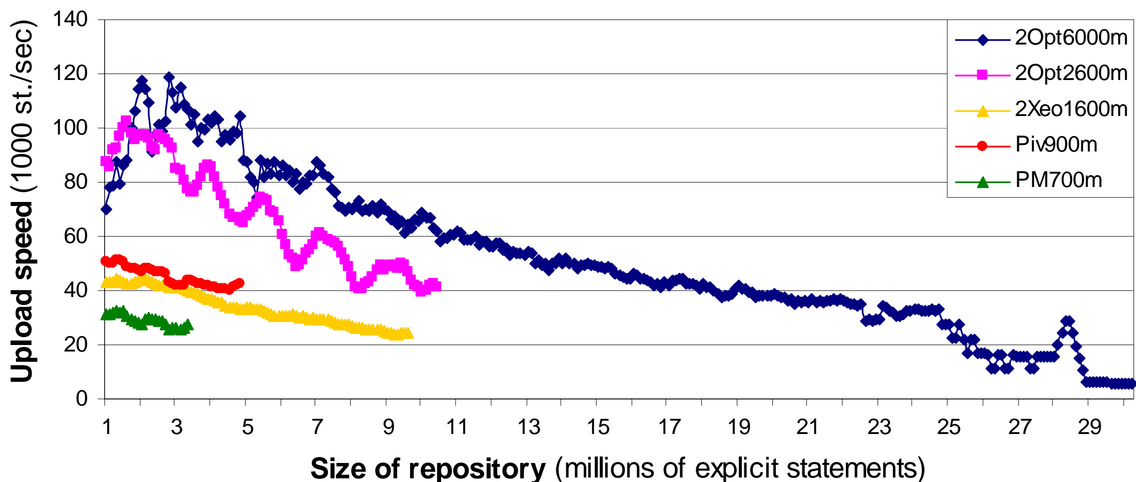

Figure 2. Upload and Inference Speed

- The maximum size of the repository varies from a few million of statements on a notebook to 30 million statements on a server with 6GB of RAM. As it was expected, the 64-bit Java virtual machine requires a bit more memory for the same size of the repository – this explains why 2Opt2600 scales just a million of statements more than 2Xeo1600.
- The query evaluation time for relatively simple queries with fixed result set (Q1) is almost constant, i.e. it almost does not grow with the size of the repository; see Figure 3.
- The time for the query evaluation for complex query (Q2) grows up in a linear dependency to the size of the repository and the size of the result set. It starts at tens of milliseconds, when the repository contains only few millions of statements, and grows up to few seconds when the repository gets bigger. Because all the query results are fetched, the total time for the query is affected by the size of the result, which grows linearly with the size of the repository, to reach tens of thousands of results – see Figure 4.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

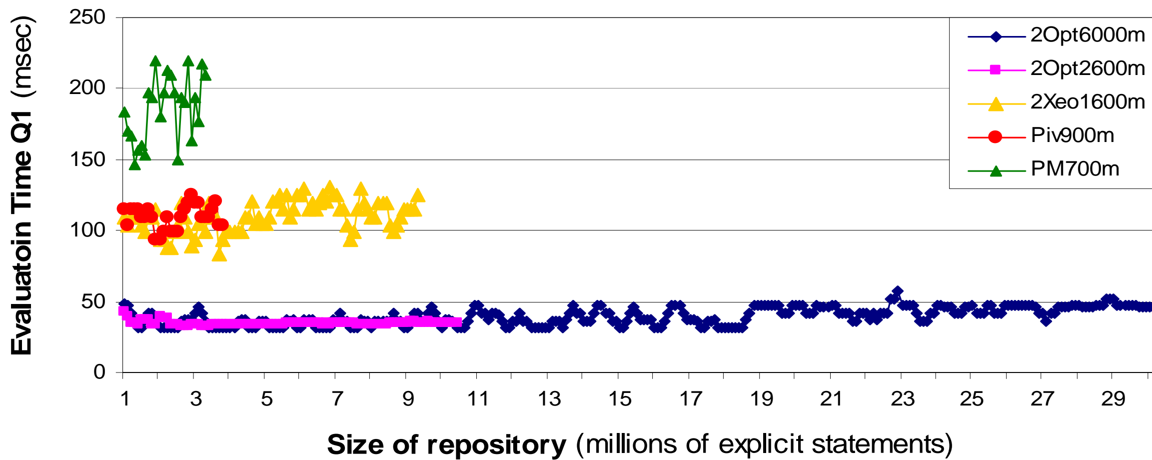

Figure 3. Query Evaluation Time (Q1)

- The delete operation is relatively slow, as it was expected, because of the straightforward invalidation of the inferred closure - with respect to the size of the repository, the time necessary to finalize the delete transaction on machine Piv900m varies in the following manner: 19 sec. for 1.5M st.; 31 sec. for 2.5M st.; 43 sec. for 3.5M st. In other words, it grows linearly with about 10 sec. for each million of statements added to the repository.

The results of the evaluation of a previous version of OWLIM (v.2.0) are published in [11] together with a more comprehensive description of the City benchmark platform.

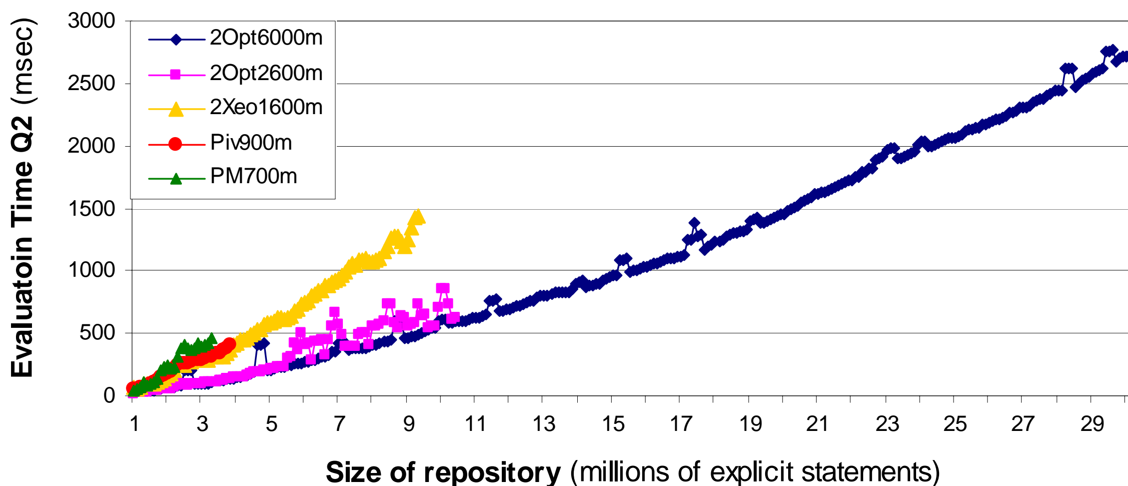

Figure 4. Query Evaluation and Result Fetching Time (Q2)

9.2 LUBM Benchmark

The Lehigh University evaluation, [7], 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

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

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). The smallest set is 8MB and the largest (for 50 universities) is 583MB, described in 1000 OWL files, with a total of about 6 million statements.

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 OWLIM distribution and documented in [13]. The results of the LUBM evaluation of the default configuration of OWLIM can be summarized as follows:

- On a desktop machine (Piv900) OWLIM loads the LUBM(50,0) dataset in 383 sec, i.e. about 6 min. According to the results published in [7], there is just one other system, that can load the benchmark but it does so in more than 12 hours!
- All the 14 queries are answered correctly. The query evaluation times for LUBM (50,0) are given in Table 2, together with the size of the results sets (column "Res.#").
- 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 2. LUBM(50,0) Query Evaluation Time for OWLIM

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			

9.3 OWLIM Performance Analysis

Several tests have been conducted in order to measure the effect of varying different parameters and configurations on the performance of OWLIM. Unless indicated otherwise, the analysis reported in this section have been performed on 2opt6000 machine (dual-Opteron, Windows 2003 Server 64-bit). We measure the time for loading LUBM(50,0) dataset (about 6M explicit st.; see section 9.2). Loading, in the case of OWLIM, includes the following operations:

- Parsing the XML files;
- Inference, the inferred closure is calculated through forward-chaining and total materialization. This operation is irrelevant when the **`empty`** rule-set is selected in order to force OWLIM to act as a plain RDF store;
- Persistence (unless it is switched on) of all the data;

Figure 5 combines information about different index sizes (parameter **`indexSize`**) and different Java virtual machines. Relative performance is also provided, with respect to a "base" configuration, running a 32-bit version of JDK 1.5.0_05 with the default OWLIM index size: 4 million entries. Each million of index entries requires additional 16 MB of RAM. As expected, larger index sizes lead to better performance. It seems that critical for the performance on

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

LUBM(50,0) is the border line between 1M and 2M index entries. For repositories with this size, index sizes larger than the default setting seem to deliver very little improvement. The 64-bit version of JDK 1.5.0 is almost 30% faster than the 32-bit one; however, it should be considered that it also consumes more RAM. JDK v. 1.4.2 is about 20% slower than v. 1.5.0.

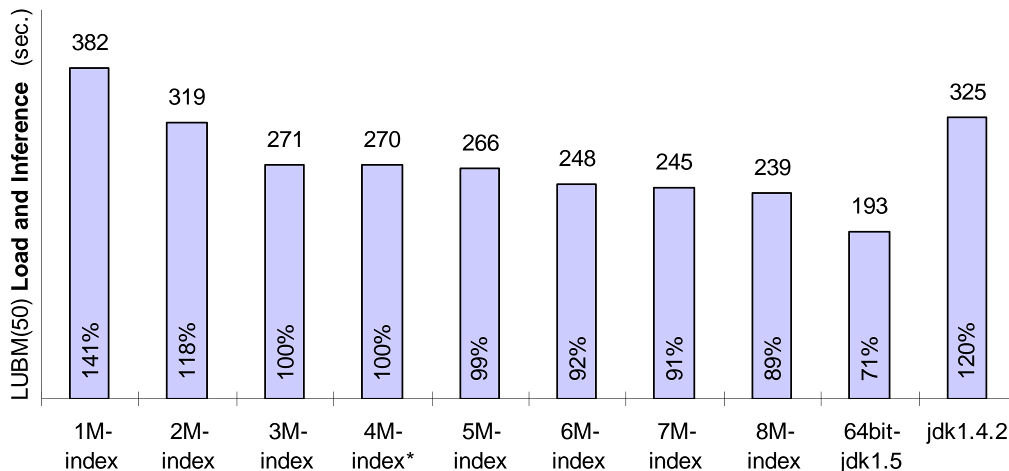

Figure 5. OWLIM Performance Dependence on Index Size and Java VM

Figure 6 demonstrates different configurations of the repository with respect to the rule-set supported (parameter **`ruleset`**), RDFS optimization (parameter **`partialRDFS`**), and persistence. It also shows the performance of the standard Sesame SAIL (**`RDFSSchemaSail,`** v.1.2.4), which supports in-memory reasoning with respect to RDFS and no sort of persistence – this is the configuration referred in [7] as Sesame-memory.

The 64-bit version of JDK 1.5.0 was used to measure all the figures given at Figure 6. Relative performance is given in percentages – 100% correspond to the performance of the default configuration of OWLIM: rule set **`owl-horst`**; **`partRDFS`** optimization switched on (indicated with pRDFS in the figure); the persistence support is also switched on. An analysis of the results follows:

- The parsing of the RDF/XML files and building of in-memory representation takes about half of the time for loading LUBM(50,0) in the default configuration of OWLIM. This is evident from the 51% relative performance (99 s.) of setup "empty, noPers.", which is version of OWLIM which does no inference or persistence.
- The persistence occupies 8-10% of the time necessary for OWLIM to load a dataset. This is evident from the 92% relative performance of setup "horst, pRDFS, noPers.", which corresponds to the default setup, but with persistence switched off. Taking that persistence in OWLIM does not depend on the complexity of the reasoning, it is more correct to say that persistence takes about 20 sec., which is 20% on top of the time for parsing and building in-memory representation.
- The default in-memory SAIL of Sesame (v.1.2.4) is more than twice slower than the "rdfs, noPers." setup of OWLIM, which provides the same functionality: standard RDFS semantics (with **`partRDFS`** optimization switched off) and no-persistence. It provides roughly the same performance as the default OWLIM configuration, but with less functionality (no persistence and no OWL support). An interesting observation for this

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

Sesame SAIL was that it performed faster on 32-bit version of JDK 1.5 (321 s.); the time required by the run on 64-bit JDK was even higher.

- The **`partRDFS`** optimization (see section 6.2) provides 20% speedup for rule set **`owl-horst`**, 14% for **`owl-max`**. This optimization makes the basic RDFS entailment in OWLIM (see rule set **`rdfs`**) only about 2% faster. The conclusion is that for simpler rule sets the **`partialRDFS`** optimization plays smaller role.

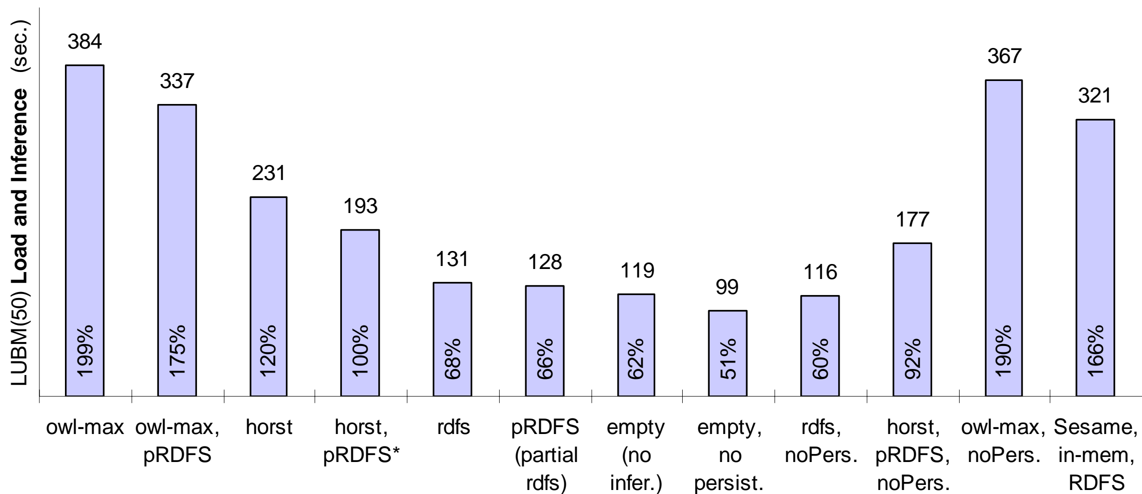

Figure 6. OWLIM Performance Dependence on Rule-set and Performance

A summary on the load time for LUBM(50,0) on different hardware follows:

- 803 s. (13 min.) on machine PM700 – a notebook with Pentium M on 1.6GHz. The index size was set to 1 million entries, to meet the memory constraints;
- 383 s. (6 min.) on machine Piv900 – a desktop machine having 3.0GHz Pentium 4 (#630: 2MB L2 cache and hyper-threading).
- 270 s. (5 min.) with 32-bit JDK on machine 2opt6000 – a dual-CPU server with Opteron 248 processors (2.0GHz and 1MB L2 cache each) and RAID 0+1 storage;
- 193 s. (3 min.) on 2opt6000 running 64-bit JDK 1.5.0.

An experiment was also conducted to detect the largest LUBM dataset which can be handled by machine 2opt6000 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.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.

10 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, 10 Feb. 2004. http://www.w3.org/TR/owl-ref/

[6] Grosof, B; Horrocks, I; Volz, R; Decker, St. (2003). *Description Logic Programs: Combining Logic Programs with Description Logic.* In Proc. of WWW2003, Budapest, May 2003.

[7] 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

[8] Hayes, P. (2004). *RDF Semantics.* W3C Recommendation 10 Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[9] 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.

[10] 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.

[11] 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.

[12] 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/

[13] 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).

[14] Motik, B., Sattler, U., Studer, R. Query Answering for OWL-DL with Rules. Journal of Web Semantics 3 (2005), pp. 41-60.

[15] Ontotext Lab. (2005). *OWLIM Tests and Benchmarks.* v2.8.2, DRAFT. http://www.ontotext.com/owlim/v2.8.2/OWLIMTests.pdf

[16] 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.

© Copyright 2005-2006, Ontotext Lab, Sirma Group Corp.