



---

## D4.4: Knowledge Store: Performance Evaluation

---

Atanas Kiryakov(Ontotext Lab, Sirma Group Corp.),  
Ivan Peikov (Ontotext Lab, Sirma Group Corp.),  
Zdravko Tashev(Ontotext Lab, Sirma Group Corp.),  
Atanas Ilchev(Ontotext Lab, Sirma Group Corp.)

### Abstract

EU-IST Specific targeted research project (STREP) IST-2004-026460 TAO  
Deliverable D4.4 (WP 4)

In this deliverable we present evaluation of the performance of the Heterogeneous Knowledge Store (HKS) – a semantic repository supporting multi-modal information retrieval. The implementation of the HKS is based on the TRREE engine used as semantic repository. The performance of TRREE is evaluated against the state of the art in the field of scalable RDF databases and lightweight inference.

---

**Keyword list:** heterogeneous knowledge, storage, evaluation, benchmarking, scalability

---

WP4 - Scalable, Heterogeneous Knowledge Stores

Document ID: **TAO/2009/D4.4/v1.0**

Nature: **Report**

Dissemination: **PU**

Contractual date of delivery: **28 Feb 09**

Actual date of delivery: **12 Jun 09**

Web links:

<http://www.tao-project.eu/researchanddevelopment/demosanddownloads/heterogeneous-knowledge-store.html>

## TAO 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-2004-026460.

### University of Sheffield

Department of Computer Science  
Regent Court, 211 Portobello St.  
Sheffield S1 4DP  
UK  
Tel: +44 114 222 1930  
Fax: +44 114 222 1810  
Contact person: Kalina Bontcheva  
E-mail: K.Bontcheva@dcs.shef.ac.uk

### Mondeca

3, cité Nollez  
75018 Paris  
France  
Tel: +33 (0) 1 44 92 35 03  
Fax: +33 (0) 1 44 92 02 59  
Contact person: Jean Delahousse  
E-mail: jean.delahousse@mondeca.com

### University of Southampton

Southampton SO17 1BJ  
UK  
Tel: +44 23 8059 8343  
Fax: +44 23 8059 2865  
Contact person: Terry Payne  
E-mail: trp@ecs.soton.ac.uk

### Sirma Group Corp., Ontotext Lab

Office Express IT Centre, 5th Floor  
135 Tsarigradsko Shosse Blvd.  
Sofia 1784  
Bulgaria  
Tel: +359 2 9768 303  
Fax: +359 2 9768 311  
Contact person: Atanas Kiryakov  
E-mail: naso@sirma.bg

### Atos Origin Sociedad Anonima Espanola

Dept Research and Innovation  
Atos Origin Spain, C/Albarracin, 25, 28037  
Madrid  
Spain  
Tel: +34 91 214 8835  
Fax: +34 91 754 3252  
Contact person: Tomás Pariente Lobo  
E-mail: tomas.parientalobo@atosresearch.eu

### Dassault Aviation SA

DGT/DPR  
78, quai Marcel Dassault  
92552 Saint-Cloud  
Cedex 300  
France  
Tel: +33 1 47 11 53 00  
Fax: +33 1 47 11 53 65  
Contact person: Farid Cerbah  
E-mail: Farid.Cerbah@dassault-aviation.com

### Jozef Stefan Institute

Department of Knowledge Technologies  
Jamova 39  
1000 Ljubljana  
Slovenia  
Tel: +386 1 477 3778  
Fax: +386 1 477 3131  
Contact person: Marko Grobelnik  
E-mail: Marko.Grobelnik@ijs.si

### List of Abbreviations

FTS	Full Text Search
HKS	Heterogeneous Knowledge Store
MSt	Million RDF statements (explicit statements are considered if there is no other indication in the context)
OWL	Web Ontology Language
RDF	Resource Description Framework
QTPR	Query time per result
SPARQL	SPARQL Protocol and RDF Query Language
TRREE	Triple Reasoning and Rule Entailment Engine
URI/L	Uniform Resource Identifier/Locator

### **Executive Summary**

One of the major prerequisites for the realization of the TAO transitioning methodology and infrastructure is the implementation of a common storage for all kinds of knowledge involved in the transitioning process. Such a repository shall offer the possibility to uniformly store content, semantic annotations, annotated Web services, ontologies and non-textual documents. Given that big volumes of data can be associated with a legacy application, the heterogeneous knowledge store shall feature highly scalable and efficient implementation.

The implemented Heterogeneous Knowledge Store provides functionality for storage, retrieval and querying data coming from: textual documents, results of content augmentation, ontologies, and semantic web services.

In this deliverable we present evaluation of the performance of the Heterogeneous Knowledge Store (HKS) – a semantic repository supporting multi-modal information retrieval. The implementation of the HKS is based on the TRREE engine used as semantic repository. The performance of TRREE is evaluated against the state of the art in the field of scalable RDF databases and lightweight inference.

The implementation of the heterogeneous knowledge store (HKS) uses the BigTRREE engine as a semantic repository that stores all kinds of heterogeneous knowledge in a single place. That is why, we have carried out an evaluation of the BigTRREE engine with respect to multiple of the most popular benchmarks and special purpose test scenarios in order to validate the following aspects of its performance:

- Loading speed (including parsing and indexing)
- Lightweight inference and materialization
- Query evaluation (standard RDBMS scenarios)
- Full-text search (FTS) queries.

The results from our evaluation have been compared with the best published results of the most popular semantic repositories and are presented in this report.

## Contents

<b>TAO Consortium</b> .....	<b>2</b>
<b>List of Abbreviations</b> .....	<b>3</b>
<b>Executive Summary</b> .....	<b>4</b>
<b>Contents</b> .....	<b>5</b>
<b>List of Tables</b> .....	<b>6</b>
<b>List of Figures</b> .....	<b>6</b>
<b>1 Introduction</b> .....	<b>7</b>
1.1 Relevance to TAO .....	7
1.1.1 Relevance to project objectives .....	7
1.1.2 Relation to other workpackages .....	8
1.2 Deliverable Outline .....	8
<b>2 Semantic Repository Tasks and Performance Factors</b> .....	<b>9</b>
2.1 Performance Dimensions .....	9
2.2 Knowledge Representation (Schema) Complexity .....	10
2.3 Reasoning Strategies Factors.....	12
<b>3 Semantic Repository Benchmarks</b> .....	<b>14</b>
<b>4 Evaluation Results</b> .....	<b>16</b>
4.1 Scalability .....	16
4.1.1 Scalability Tests with Semantic Annotations .....	18
4.2 Query Performance Evaluation .....	18
4.3 Query Evaluation - BSBM Results .....	20
4.4 Full-text Search Evaluation .....	21
<b>5 Conclusion and Future Work</b> .....	<b>23</b>
<b>Bibliography and References</b> .....	<b>24</b>

## List of Tables

Table 1: BSBM Benchmark.....	15
Table 2: Scalability evaluation.....	17
Table 3: Loading Speed.....	18
Table 4: Query Evaluation Performance for LUBM (8000), 1 billion statements .....	19
Table 5: Query Evaluation Performance with native FTS support .....	22

## List of Figures

Figure 1: OWL Layering Map .....	12
Figure 2: BSBM Results Chart .....	20
Figure 3: Query Evaluation.....	20

# 1 Introduction

One of the major prerequisites for the realization of the TAO transitioning methodology and infrastructure is the implementation of a common storage for all kinds of knowledge involved in the transitioning process. Such a repository shall offer the possibility to uniformly store content, semantic annotations, annotated Web services, ontologies and non-textual documents. Given that big volumes of data can be associated with a legacy application, the heterogeneous knowledge store shall feature highly scalable and efficient implementation.

The implemented Heterogeneous Knowledge Store provides functionality for storage, retrieval and querying data coming from: textual documents, results of content augmentation, ontologies, and semantic web services.

In this deliverable we present evaluation of the performance of the Heterogeneous Knowledge Store (HKS) – a semantic repository supporting multi-modal information retrieval. The implementation of the HKS is based on the TRREE engine used as semantic repository. The performance of TRREE is evaluated against the state of the art in the field of scalable RDF databases and lightweight inference.

The implementation of the heterogeneous knowledge store (HKS) uses the BigTRREE engine as a semantic repository that stores all kinds of heterogeneous knowledge in a single place. That is why, we have carried out an evaluation of the BigTRREE engine with respect to multiple of the most popular benchmarks and special purpose test scenarios in order to validate the following aspects of its performance:

- Loading speed (including parsing and indexing)
- Lightweight inference and materialization
- Query evaluation (standard RDBMS scenarios)
- Full-text search (FTS) queries.

The results from our evaluation have been compared with the best published results of the most popular semantic repositories and are presented in this report.

## 1.1 Relevance to TAO

An important outcome of TAO will be the semantic-based access to heterogeneous knowledge, structured data and metadata, that are associated with the legacy application or are result of the transitioning process. The Heterogeneous Knowledge Store (HKS) developed in WP4 is responsible for providing storage and retrieval of diverse types of data into a common semantic repository and for providing query functionality over that repository. In this section we briefly summarize the relevance of HKS for TAO in terms of its relation to project objectives and to the rest of the workpackages.

### *1.1.1 Relevance to project objectives*

The Heterogeneous Knowledge Store development is directly related to the second project objective of TAO – Augmentation and integration of legacy content. It supports storage of the heterogeneous knowledge associated with the legacy application being transitioned as well as with the heterogeneous results produced

## D4.4 Knowledge Store: Performance Evaluation

during the transitioning process. It will also be part of the TAO Suite and hence is also related to the objective of providing Transitioning Methodology and Infrastructure.

HKS is a very innovative semantic repository which advances the state of the art with respect to the ability to store, access and query different types of data in a highly scalable and efficient way. It also addresses the problem of combining semantic-based and full-text search over the semantic repository, which together with the unified storage of diverse data will enable enterprises to easily search and access all their knowledge.

### *1.1.2 Relation to other workpackages*

HKS provides storage of all types of data used as input to and output of all TAO components. For this reason its development was carried out in coordination with all technical workpackages. Thus, HKS is designed to meet the requirements and to easily exchange data with the tools produced in WP2 and WP3 and to be seamlessly integrated within the TAO Suite. Additionally, TAO use cases will also use HKS in their case study implementations.

In more detail, the HKS is used as storage for the following data:

- Domain and application ontologies generated by WP2 tools;
- WP3 augmentation artefacts – semantic annotations, textual content, knowledge base instances;
- WP5 stores inputs and outputs of the transitioning process, and interacts with the HKS throughout that process, storing and retrieving intermediate results produced by the other tools.
- Use case data – ontologies and semantic annotations, created in WP6 and WP7.

## **1.2 Deliverable Outline**

This deliverable is structured as follows:

- Section 2 discusses some factors and tasks that need to be considered in the evaluation of a semantic repository,
- Section 3 briefly outlines the benchmarks used in our evaluation,
- Section 4 presents the results from our evaluation,
- Section 5 concludes this report.

## 2 Semantic Repository Tasks and Performance Factors

The performance of a semantic repository (SR) has to be measured at least for the following tasks:

- **Data loading**, including storing and indexing of both instance data and ontologies. The performance depends on several factors:
  - **Materialization** – whether and to what extent forward-chaining is performed at load time;
  - **Complexity of the data model** – some SR employ extended RDF data models, e.g. including support for named graphs. Richer data-models are more “expensive” to build and maintain.
- **Query evaluation**. There are several factors which affect the time and memory space, or more generally, the computing resources, necessary for this task:
  - **Deduction** – whether and to what extent backward-chaining is involved;
  - **Size of the result-set** – fetching large result-sets can take considerable time.
  - **Query complexity** – the number of the constraints (e.g. triple-pattern joins), the semantics of the query (e.g. negation-related clauses), the usage of operators which are tough to support through indexing (e.g. LIKE).

In search for a meaningful measure of the evaluation speed across a set of diverse queries, we introduce a metric, called query-time-per-result (QTPR). It is calculated as query evaluation time divided by the number of the results. Although QTPR is not the perfect query performance measure, we claim that it is more useful than the query evaluation time, taken as it is. At least, it allows for accounting the fetching time, which has a serious impact for large result sets. It also allows a meaningful average measure across query sets where the number of results varies in orders of magnitude.

Another measure of query performance could be the response time, in the sense of time elapsed before the first result is returned. There is no reference to this measure in the rest of this analysis because such data is not generated by the popular benchmark frameworks and because it is not publicly available. Still, this measure is highly relevant to a wide class of applications and should be considered in the course of performance evaluation of semantic repositories.

### 2.1 Performance Dimensions

Reasoning performance targets are defined in terms of speed, in the sense of either throughput or response time. There are several parameters which affect the speed of a semantic repository for both loading and query evaluation:

- **Scale** – the size of the repository in terms of number of facts/triples. For engines using forward-chaining and materialization, the volume of the data they have to play with includes the inferred facts/triples. Note that the

RDF/OWL representation of Wordnet<sup>1</sup> expands after materialization from 1.9 million statements to 8.2 million;

- **Schema and data reasoning complexity** – the complexity of the ontology/logical language, the specific ontology (or schema), and the dataset. E.g. a highly interconnected dataset, with long chains of transitive properties, can appear much more challenging for reasoning compared to another dataset, even when both are encoded against one and the same ontology;
- **Hardware and software setup** – the performance can vary considerably depending on the version/configuration of the compiler/virtual machine, the operating system, the configuration of the SR and the hardware configuration.

The above analysis demonstrates that one should be realistic about the expectations regarding the utility of a simple target for the performance of a SR. Each specific benchmark provides information about the performance of an engine for a specific combination of parameters. It presents a low-dimensional section (the benchmark results) of the multi-dimensional “body” of the SR performance. A benchmark designed to measure the performance of a SR, dealing with single user’s calendar and contact data on a smart phone, is not likely to be useful for evaluating another SR, which should allow hundreds of users to investigate billions of facts in the life science domain. The same holds for the RDBMS – one can hardly define a single simple performance target to measure their performance on. It took decades for the RDBMS community to come up with standard benchmarks like TPC (<http://www.tpc.org/>). Yet, the major RDBMS vendors have the policy to avoid competitive performance benchmarks. Today, TPC is primarily used by hardware and OS vendors to compare the performance and the cost efficiency of server configurations.

### 2.2 Knowledge Representation (Schema) Complexity

Even though RDFS can be considered a very simple Knowledge Representation (KR) language, it is a challenging task to implement a repository for it, especially such that provides performance and scalability comparable to those of relational database management systems (RDBMS). Going up the “layers” of the Semantic Web specifications stack, the challenges for the repository engineers become more serious. Even the simplest official dialect of OWL (OWL Lite) is based on Description Logic (DL), the complexity of which renders sound and complete reasoning against large knowledge bases (KB) theoretically impossible. Furthermore, the semantics of OWL Lite and OWL DL are incompatible with that of RDF(S), see [4]. This causes lack of “backward compatibility”. Imagine the situation when an application, using RDFS schemata and RDFS-compliant repository, should be “upgraded” to OWL. The evolution should start with replacement of RDFS schemata with OWL ontologies and adoption of a repository supporting (the corresponding part of) OWL. Even the most direct translation (re-labeling the **rdfs:Class** to **owl:Class**) of the schema, can lead to different inference and to inconsistencies.

---

<sup>1</sup> Wordnet is the most popular lexical knowledge base, <http://www.w3.org/TR/wordnet-rdf/>. Results on loading its RDF/OWL representation in OWLIM are presented in **Error! Reference source not found.**

## D4.4 Knowledge Store: Performance Evaluation

Logic programming (LP) is a common name used for rule-based logical formalisms such as PROLOG, Datalog, and F-Logic. OWL DLP is a non-standard dialect, offering a promising compromise between expressive power, efficient reasoning, and RDFS compatibility. OWL DLP is defined in [4] 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, in comparison with the OWL Lite and OWL DL dialects. Still, this mapping can only be achieved through the enforcement of some additional modelling constraints and transformations. A broad collection of resources related to OWL DLP can be found at <http://logic.aifb.uni-karlsruhe.de/>.

In [14], 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, [6], 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.

Horst extends and modifies the D-entailment rules from [6] in two steps as follows: D\*-entailment adds entailment support for literal data-types; pD\*-entailment adds rules which provide partial support for some OWL primitives, namely: **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 [14], it has several 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 the “unique name” assumption;
- Unlike the DL-based rule languages, like SWRL, [7] and [9], 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 [14].

We have observed that most of the languages supported by the contemporary scalable semantic repositories are very similar to OWL-Horst, even when additional or alternative rules are used. Although additional rules make entailment more expensive, “safe” extensions raise the complexity by a constant. According to [14], a sufficient condition for remaining in the same class of complexity is that the rules should not

introduce new blank nodes in the knowledge base. Our experience shows, that for real-world OWL ontologies and datasets, it is sufficient to enforce the above conditions only with respect to the rules, dealing with ABox reasoning (e.g. with instances). For instance, introducing a new blank node, which represents an auxiliary OWL restriction (or class), does not actually lead to exponential growth of the inferred facts.

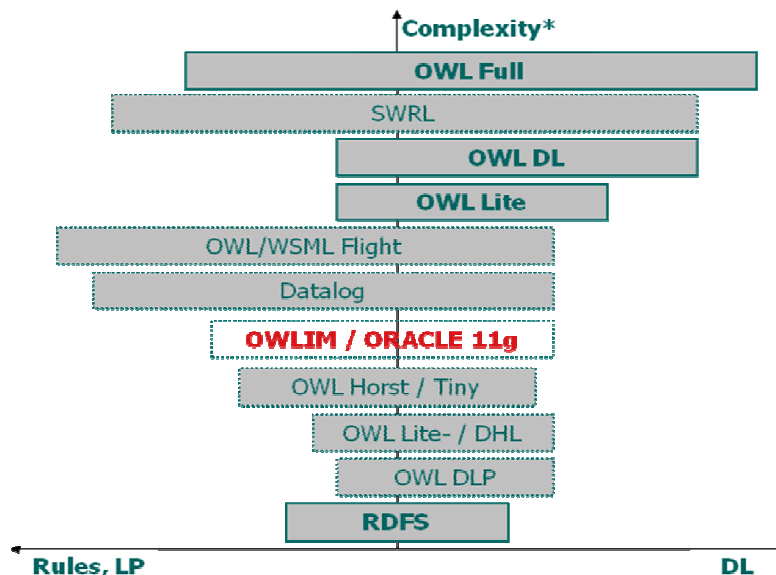


Figure 1: OWL Layering Map

Figure 1 presents a simplified map of the expressivity or complexity of a number of OWL-related languages, as well as their bias towards DL and LP semantics. The figure provides rough evidence about the expressivity of the languages, based on the complexity of entailment algorithms for them. Direct comparison between these different languages is impossible in many of the cases. For instance, Datalog is not simpler than OWL DL, it introduces a different type of complexity.

### 2.3 Reasoning Strategies Factors

Nowadays, many of the scalable semantic repositories perform reasoning based (more or less directly) on logical entailment rules. 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*<sup>2</sup>; 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 (bindings for the free variables), 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 knowledge base (KB) or can be proven through further recursive transformations.

<sup>2</sup> *Inferred closure* is defined here 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.

## D4.4 Knowledge Store: Performance Evaluation

Both strategies have well-known strong and weak points. Hybrid strategies are also possible and have proven to be efficient in many contexts.

Let us imagine a repository which performs total forward-chaining, i.e. after update to the KB the inferred closure is updated and kept available for query evaluation and retrieval. This strategy is known as *materialization*. In order to avoid ambiguity with various partial materialization approaches, let us call such inference strategy *total materialization*.

The principle advantages and disadvantages of the total materialization are discussed at length in [1] here we provide a brief summary of them:

- Upload/storage/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. Query evaluation becomes computationally comparable to the same task for relation database management systems (RDBMS).

Apart from the principle advantages and disadvantages of the reasoning strategies, their applicability is often pre-determined by the complexity of the semantics (ontology language) that has to be supported. Imagine a dataset, where large groups of individuals are interconnected through a transitive and symmetric property, e.g. fellow-citizen (as in [8]). If each citizen of a city with 100 thousand citizens is connected explicitly, say, with two others, there will be 200 thousand explicit fellow-citizen statements in the dataset. However, the inferred closure will contain about 10 billion statements! Obviously, total materialization can face troubles even with tractable logical formalisms.

### 3 Semantic Repository Benchmarks

Below some benchmarks and datasets, most commonly used as measuring sticks for the performance of semantic repositories, are presented.

The Lehigh University Benchmark (LUBM) is an outstanding benchmark for OWL repository scalability, defined in [3]. It employs synthetically generated datasets using a fixed OWL ontology of university organizations. The complexity of the language constructs used is between the one of OWL Horst and OWL DLP, as shown on Figure 1. The LUBM benchmark defines 14 queries that are used to check the query evaluation correctness and speed of repositories that have loaded a given dataset. The biggest standard dataset is LUBM (50) (i.e. it contains synthetic data for 50 universities); its size is 6.8 million explicit statements, and if written to the hard disk (in 1000 RDF/XML files), it takes 600Mbytes. For the purposes of scalability measurements many groups have used the LUBM generator to create bigger datasets, e.g. LUBM (1000) and LUBM (8000) which contain respectively 133 and 1086 million explicit statements.

UniProt<sup>3</sup> (Universal Protein Resource) is the world's most comprehensive and most popular database of information on proteins, created by joining the information contained in several other resources (Swiss-Prot, TrEMBL, and PIR). UniProt RDF, [13] is an RDF representation of the database with respect to OWL ontology, expressed in a sub-language of OWL Lite. As it represents one of the largest real-world datasets represented in RDF and OWL, managing UniProt it is often used as benchmark for scalability and reasoning capabilities of semantic repositories. While in the summer of 2007 the size of the RDF representation of UNIPROT was about 384M statements, in the beginning of 2008 its size approaches a billion of explicit statements. The RDF graph defined in the UNIPROT ontology is highly interconnected, which has significant impact on the reasoning speed of semantic repositories.

Over the last year the vision for the Semantic Web as a web of structured data started materializing in the form of the so-called “linked data”: an initiative for publishing RDF data on the web, following some principles which allow exploration/navigation and interlinking. The Linking Open Data project<sup>4</sup> (LOD) has already collected over 40 datasets which are interlinked between each other and contain in total above 3 billion RDF statements. The central dataset within LOD is DBPedia<sup>5</sup> - an RDF dataset derived through extraction of structured information from Wikipedia<sup>6</sup>. It currently contains about 218 million explicit statements describing and interconnecting about 2 million entities. DBPedia is also very well interconnected with several other datasets, e.g. W3C's representation of Wordnet in RDF(S)/OWL. As in the case of UNIPROT, it is not a benchmark, but rather a challenging non-synthetic datasets which could be used for benchmarking purposes.

---

<sup>3</sup> [www.uniprot.org/](http://www.uniprot.org/)

<sup>4</sup> <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>

<sup>5</sup> <http://wiki.dbpedia.org/>

<sup>6</sup> <http://www.wikipedia.org/>, the most popular open web encyclopaedia.

## D4.4 Knowledge Store: Performance Evaluation

The BSBM benchmark<sup>7</sup> is designed to compare triple store implementation that exposes a SPARQL endpoint with realistic workloads of e-commerce use case motivated queries. The queries simulate the navigation and exploring of products by one or multiple end-users. The benchmark supports the generation different scales and data sets:

<b>Number of Producers</b>	<b>14</b>	<b>60</b>	<b>1422</b>	<b>5,618</b>
<b>Number of Product Features</b>	2,860	4,745	23,833	47,884
<b>Number of Product Types</b>	55	151	731	2011
<b>Number of Vendors</b>	8	34	722	2,854
<b>Number of Offers</b>	13,320	55,700	1,416,240	5,696,520
<b>Number of Reviewers</b>	339	1432	36,249	146,054
<b>Number of Reviews</b>	6,660	27,850	708,120	2,848,260
<b>Total Number of Instances</b>	23,922	92,757	2,258,129	9,034,027
<b>Number of RDF Statements</b>	250K	1M	25M	100M

**Table 1: BSBM Benchmark**

An important quality of the benchmark is to cover a complete scenario of usage of the functionality of the tool. For instance, passing LUBM involves the following activities of a semantic repository:

- Parsing of RDF/XML;
- Storage/persistence;
- Indexing;
- Reasoning;
- Query evaluation.

The performance is measured for loading and for query evaluation – two tasks which provide indication about the performance with regard to all the activities and thus provide a consistent picture about the efficiency of specific tool or approach. Without such complete picture, there is no empirical proof for the performance with regard to some of the activities. For instance, inference can take place both at load or at query time or to be split into different phases between those two modalities of usage. There is no perfect inference schema or configuration for SR – different approaches provide optimal results in different situations; forward-chaining on top of very dynamic dataset with large “inferred closure” is suboptimal. The same applies to indexing – loading can be very efficient if little or no indices are created and stored; this however will have impact on the query performance.

When evaluating the performance of a SR or a specific configuration, one should pay attention mainly to the results from full-cycle-benchmarks, which allow consistent interpretation of its advantages and the disadvantages. If for instance, only the results of loading an LUBM dataset are presented, without query performance evaluation, there is no trustworthy indication about the complexity of the inference and indexing involved in this task.

---

<sup>7</sup> <http://www4.wiwi.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/index.html>

## 4 Evaluation Results

This section presents the results from the evaluation carried out for the TRREE semantic repository, used by HKS. Note that for some of the experiments we used OWLIM as interface to the TRREE in order to minimize the influence of interface complexity over the results.

### 4.1 Scalability

Table 2 presents the loading time and speed of the engines for the different datasets. For each run we provide the following data:

- **Scale** and **Time**, in number of explicit statements loaded and hours, respectively;
- **Speed**: average speed of loading, in thousands of statements per second;
- **Overall complexity index**: average of the Forward-Chaining and the Parsing and Indexing Complexity indices. It represent a combined measure for the complexity of the loading task;
- **Forward-chaining semantics**: what language is used for forward-chaining and materialization during the loading, if any;
- **Forward-chaining complexity-index**: our subjective estimate about the complexity of the reasoning during this run;
- **Parsing & Indexing Complexity**: our subjective estimate about the complexity of the loading tasks that are not related to indexing. It includes the specifics of the concrete run and system, e.g.: rich RDF model, full-text search indexing, etc.

## D4.4 Knowledge Store: Performance Evaluation

Semantic Repository	Dataset / Run	Scale (mill. ex.st.)	Speed (KSt./sec.)	Overall Complexity Rate	Loading Time (hours)	Forward-Chaining Semantics	Materialization Complexity Rate	Parsing & Indexing Complexity
BigOWLIM 3.1	LUBM(1k)	138	25,617	15	1.50	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(1k)	138	76,159	5	0.50	none	0	10
BigOWLIM 3.1	UNIPROT	1,150	15,212	20	21.00	OWL-Horst	30	10
BigOWLIM 3.1	PIKB	1,466	9,694	28	42.00	OWL-Horst	45	10
BigOWLIM 3.1	LDSR	357	14,167	28	7.00	OWL-max	45	10
BigOWLIM 3.1	LUBM(8k)	1,068	42,381	9	7.00	RDFS	10	7
BigOWLIM 3.1	LUBM(8k)	1,068	20,631	15	14.38	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(8k)	1,068	66,196	5	4.48	none	0	10
BigOWLIM 3.0	LUBM(20k)	2,760	44,516	5	17.22	none	0	10
BigOWLIM 3.0	LUBM(50k)	6,675	43,914	5	42.22	none	0	10
BigOWLIM 3.1	LUBM(16k)	2,136	13,267	15	44.73	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(24k)	3,204	13,039	15	68.26	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(32k)	4,272	12,772	15	92.92	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(40k)	5,340	12,559	15	118.11	OWL-Horst	20	10
AllegroGraph 3.0 Federated	LUBM(8k)	1,107	37,273	5	8.25	none	0	10
AllegroGraph 3.2	LUBM(8k)	1,107	13,667	5	22.50	none	0	10
Openlink Virtuoso, 5.0	LUBM(8k)	1,068	36,900	5	8.04	none	0	10
ORACLE 11g	LUBM(1k)	138	3,576	15	10.72	OWL-Prime	20	10
ORACLE 11g	LUBM(8k)	1,068	8,725	5	34.00	none	0	10
ORACLE 11g	LUBM(8k)	1,068	3,333	15	89.00	OWL-Prime	20	10
DAML DB	LUBM(8k)	850	10,266	14	23.00	OWL-Horst	20	7
DAML DB	LUBM(8k)	220	6,111	14	10.00	OWL-Horst	20	7
Jena TDB	UNIPROT v.13.3	1,516	11,698	5	36.00	none	0	10
BigData - cluster	LUBM(8k)	1,155	51,041	5	6.28	none	0	10
BigData - cluster	LUBM(8k)	1,155	32,201	10	9.96	RDFS	10	10
BigData - clusrer	LUBM(?)	5,000	138,889	5	10.00	none	0	10
BigData - clusrer	LUBM(?)	9,000	80,645	5	31.00	none	0	10
BigData - clusrer	LUBM(?)	10,400	61,466	5	47.00	none	0	10

**Table 2: Scalability evaluation**

## D4.4 Knowledge Store: Performance Evaluation

### 4.1.1 Scalability Tests with Semantic Annotations

In this section we briefly present the results obtained by importing the GATE use case corpus, provided by WP6, into the Heterogeneous Knowledge Store. These results are obtained on a desktop machine featuring Intel Core Duo @ 2.20GHz 1,5G RAM.

The parameters of this evaluation are as follows:

- NumberOfStatements = 58233396
- NumberOfExplicitStatements = 58233320
- NumberOfEntities = 19839212
- Total Annotations: 3561312
- Total Time: 24486347ms

Intermediate results and snapshots show that at the beginning the loading speed is 2annotations/ms, but then it gradually degrades.

Annotations	Time in ms
1086	1042
4229	1933
5369	2462
1205	567
7933	3152
697	25072

Table 3: Loading Speed

## 4.2 Query Performance Evaluation

Query evaluation speed is the second major performance criteria analyzed here. There is very little public information available about query performance on datasets with close to one billion statements:

- The YARS2 data in [5] – they were excluded for the reasons given above;
- The results of BigOWLIM, evaluating the LUBM(8000) queries in [11]
- The results of DAML DB for the LUBM (8000), published in [12] – relevant, but they lack concrete numbers, rather provide graphic on a logarithmic scale instead.

Table 4 presents the results published in [11], which compare the query evaluation times of two versions of BigOWLIM: 0.9.5 and 0.9.6/2.0. It presents the number of results returned for each of the queries and the query evaluation times of each version for each of the queries. We can see that the number of the results returned varies between 83 millions and 4, i.e. there is a difference of some 7 orders of magnitude. Obviously, the criteria for fast evaluation between queries number 1 and 6 are different. For this purpose we adopt the QTPR (query-time-per-result) metric.

#### D4.4 Knowledge Store: Performance Evaluation

Query No	Number of results	BigOWLIM 0.9.5		BigOWLIM 0.9.6/2.0	
		Time (ms)	QTPR	Time (ms)	QTPR
1	4	3 962	991	100	25
2	2 528	1 144 726	453	181 015	72
3	6	24 506	4 084	100	17
4	34	123 978	3 646	182	5
5	719	31 018	43	92	0
6	83 557 706	169 375	0	106 216	0
7	67	123 806	1 848	154	2
8	7 790	42 514	5	914	0
9	2 178 420	2 318 412	1	738 032	0
10	4	37 704	9 426	102	26
11	224	413	2	75	0
12	15	1 317	88	453	30
13	37 118	22 536	1	16 061	0
14	63 400 587	383 261	0	83 792	0
<b>Average</b>			<b>1471</b>		<b>13</b>

**Table 4: Query Evaluation Performance for LUBM (8000), 1 billion statements**

The average QTPR of BigOWLIM ver. 0.9.6/2.0 on LUBM for dataset of one billion statements is about 13 milliseconds. The result of the older version (0.9.5) of the same engine is over a hundred times lower. The major difference between the two versions is that the newer one makes better use of statistics for the sake of query plan optimizations, similar to those employed in the RDBMS. The difference in the results demonstrates an expected fact, namely, that query optimization is crucial for datasets of this size. As this type of query optimizations are practically impossible for a reasoning engine, which performs backward-chaining in the process of query evaluation, the above result provides evidence that such engines will face principle problems with their query evaluation performance on large datasets.

In our interpretation the query evaluation results given in [12] suggest the following query times of DAML DB over an 850 million statements subset of LUBM (8000):

- Query 1 time: ~1 hour; QTPR: 15 min. Obviously DAML DB's performance here can be a subject of improvement.
- Query 14 time: ~6 hours; QTPR ~0.3 msec. Although this is much slower than the 83 sec. needed by BigOWLIM, the QTPR is good;
- Query 9 time: ~1 hour; QTPR ~2 msec. Again, the QTPR is good.

Considering that the current version of BigOWLIM works with "simple" RDF model (e.g. without support for Named Graphs), one can expect that a well tuned engine can achieve at present speeds in the range of 10-30 milliseconds QTPR on LUBM(8000), depending on its other features.

### 4.3 Query Evaluation - BSBM Results

Berlin SPARQL Benchmark, [3], evaluates the performance of query engines in e-commerce use case: searching products and navigating through related information. Randomized “query mixes” (of 25 queries each) are evaluated continuously through client application which communicates with the repository through SPARQL endpoint. The benchmark allows evaluation with respect to changing sizes of the dataset and different number of simultaneous users. Recent results from evaluation of few of the most popular engines are published in [2]. Figure 2 and Figure 3 provides a comparison between results from our evaluation of BigOWLIM 3.1 with results for other systems from [2] regarding query evaluation with growing number of simultaneous clients (1, 4, 8) against 25-million statement version of the BSBM dataset.

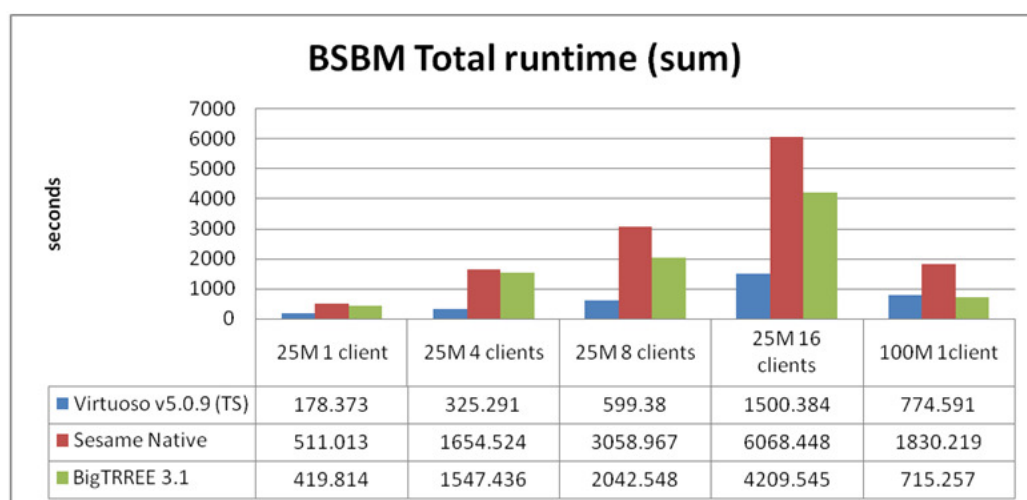


Figure 2: BSBM Results Chart

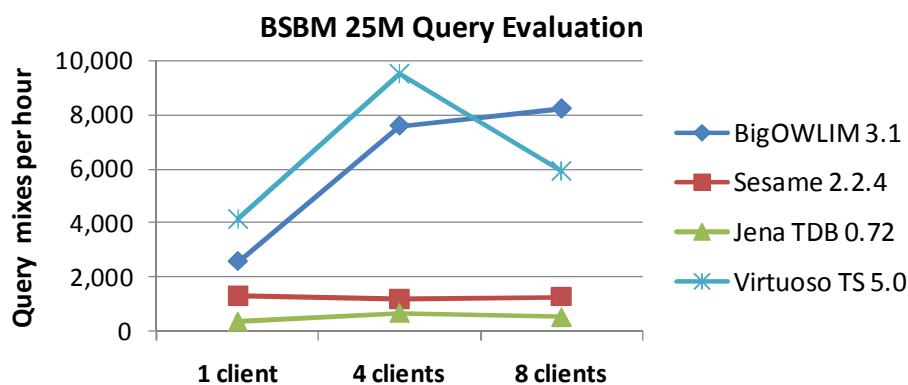


Figure 3: Query Evaluation

Given larger number of simultaneous clients, BigOWLIM and Virtuoso can deliver considerably larger throughput.

#### 4.4 Full-text Search Evaluation

The full-text search (FTS) mechanism of the TRREE storage engine provides very efficient way to search for one or more tokens inside all repository literals. For performance purposes a FTS index is maintained. The index contains a trie data structure of all tokens occurring inside the repository literals and maps them to lists of the container literals. Such a data structure can answer very efficiently questions like “*which literals in this repository contain such a token (or a prefix of it)?*”. A token lookup in the FTS index only depends linearly on the size of the token and is independent of the repository size. Thus, the FTS index should be preferred when possible instead of SPARQL's built-in REGEX function (the REGEX searching approach tends to depend linearly on the number of literals in the repository which is generally a huge number).

The TRREE engine implements FTS queries through a number of system predicates. For example, a triple pattern like

```
<hello:world> <http://www.ontotext.com/exactMatch> ?o
```

would bind the ?o variable to all literals that contain both the words “hello” and “world”. Other FTS system predicates (apart from **exactMatch**) include **prefixMatch** and the respective case-insensitive versions – **matchIgnoreCase** and **prefixMatchIgnoreCase**.

In order to evaluate the FTS performance in comparison to the standard SPARQL's REGEX performance we devised 4 queries of increasing complexity, each in 2 versions – the first using TRREE's FTS predicates and the second using SPARQL's REGEX filter. The queries are designed for execution against the WordNet dataset and use the following prefixes for brevity:

```
PREFIX onto: <http://www.ontotext.com/>
PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>
```

Follows the list of query versions:

##### Query 1: Simple case-insensitive token search

```
1. SELECT * WHERE {
    ?s ?p ?o .
    <district:> onto:matchIgnoreCase ?o
}
2. SELECT * WHERE {
    ?s ?p ?o
    FILTER(REGEX(STR(?o),
        "(^[^a-z0-9_])district($[^a-z0-9_])", "i"))
}
```

##### Query 2: Case-insensitive FTS query for two coinciding words (e.g. "new" and "york")

```
1. SELECT * WHERE {
    ?s ?p ?o .
    FILTER(REGEX(STR(?o),
```

## D4.4 Knowledge Store: Performance Evaluation

```
        "(^|[a-z0-9_])new($|[a-z0-9_])", "i")) .
    FILTER(REGEX(STR(?o),
        "(^|[a-z0-9_])york($|[a-z0-9_])", "i")) .
    }
2. SELECT * WHERE {
    ?s ?p ?o .
    <new:york> onto:matchIgnoreCase ?o
}
```

### Query 3: Case-sensitive prefix search for a common prefix (e.g. "wh")

```
1. SELECT * WHERE {
    ?s ?p ?o .
    FILTER(REGEX(STR(?o), "(^|[A-Za-z0-9_])wh")) .
}
2. SELECT * WHERE {
    ?s ?p ?o .
    <wh:> onto:prefixMatch ?o
}
```

### Query 4: More complicated query - join of several triple patterns with case-insensitive prefix match

```
1. SELECT ?aSynset WHERE {
    ?aSynset wn:containsWordSense ?aWordSense .
    ?aWordSense wn:word ?aWord .
    ?aWord wn:lexicalForm ?aForm .
    FILTER(REGEX(STR(?aForm),
        "(^|[A-Za-z0-9_])bank", "i")) .
}
2. SELECT ?aSynset WHERE {
    ?aSynset wn:containsWordSense ?aWordSense .
    ?aWordSense wn:word ?aWord .
    ?aWord wn:lexicalForm ?aForm .
    <bank:> onto:prefixMatchIgnoreCase ?aForm .
}
```

The WordNet dataset contains 841K entities and 2.8M statements (1.9M explicit and 930K inferred). The tests were performed on an Intel i7 2.93GHz quad-core Linux machine with 8G RAM. The following table summarizes the results:

Query	# of results	REGEX time (ms)	FTS time (ms)
Query 1	173	11538	13
Query 2	162	12051	16
Query 3	19092	11358	216
Query 4	136	1686	44

Table 5: Query Evaluation Performance with native FTS support

The results indeed demonstrate that the more significant the token search is to the query, the better the FTS version of the query performs than the REGEX one. Even for a comparatively small dataset (such as WordNet) FTS outperforms the REGEX approach by a factor of 1000. As the REGEX approach depends on the size of the dataset, larger dataset are expected to experience even larger factors.

## 5 Conclusion and Future Work

The implementation of the heterogeneous knowledge store (HKS), namely the latest generation of the BigTRREE engine has been evaluated with respect to multiple of the most popular benchmarks and special purpose test scenarios in order to validate the following aspects of its performance:

- Loading speed (including parsing and indexing)
- Lightweight inference and materialization
- Query evaluation (standard RDBMS scenarios)
- Full-text search (FTS) queries.

The results have been compared with the best published results of the most popular semantic repositories and can be summarized as follows:

- HKS can deal with up to 10 billion statements on a commodity database server with cost in the range of \$10 000;
- The new full-text indexing allows speed of FTS in the range of 10 to 1000 times;
- Based on the published results, HKS has no competition with respect to reasoning with more than 2 billion statements.

HKS is already part of the MIMIR engine – the next generation of universal back-end repository that is positioned as standalone product. It is also in the basis of the plans for development of the KIM semantic annotation and search platform. MIMIR is already used as data layer of the LarKC project, aiming at web-scale reasoning.

## Bibliography and References

- [1] 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>
- [2] Fokoue, A; Kershenbaum, A; Ma, L; Schonberg, E; Srinivas, K. (IBM) (2006). The Summary Abox: Cutting Ontologies Down to Size. In: Proc. ISWC-06. Volume 4273 of LNCS.
- [3] 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, pp. 158-182. <http://www.websemanticsjournal.org/ps/pub/2005-16>
- [4] Grosz, B; Horrocks, I; Volz, R; Decker, St. (2003). *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WWW2003, Budapest, May 2003.
- [5] Harth, A; Umbrich, J; Hogan, A; Decker, S. (2007) YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In Proc. ISWC 2007. <http://sw.deri.org/2007/02/swsepaper/iswc2007.pdf>
- [6] Hayes, P. (2004). *RDF Semantics*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
- [7] Horrocks, I., Patel-Schneider, P. F., Bechhofer, S., Tsarkov, D. (2005) *OWL Rules: A Proposal and Prototype Implementation*. [Journal of Web Semantics 3 \(2005\), pp. 23-40.](http://www.websemanticsjournal.org/ps/pub/2005-16)
- [8] Ma, L; Yang, Y; Qiu, Z; Xie, G; Pan, Y. (2006) *Towards A Complete OWL Ontology Benchmark*. In Proc. of the 3rd European Semantic Web Conference (ESWC 2006). Budva (Montenegro).
- [9] Motik, B; Sattler, U; Studer, R. (2005) *Query Answering for OWL-DL with Rules*. [Journal of Web Semantics 3 \(2005\), pp. 41-60.](http://www.websemanticsjournal.org/ps/pub/2005-16)
- [10] Ognyanov D., Kiryakov A., Momchev V., Velkov R., Model and Specification for Distributed Knowledge Stores, October 2006, TAO project deliverable D4.1, <http://www.tao-project.eu/resources/publicdeliverables/d4-1-final.pdf>
- [11] Ontotext Lab. (2008) *OWLIM – Pragmatic OWL Semantic Repository*. Presentation. <http://www.ontotext.com/owlim/OWLIMPres.pdf> (as of 18 Jun, 2008).
- [12] Ruhloff, K; Dean, M; Emmons, I; Ryder, D; Sumner, J. (2007). *An Evaluation of Triple-Store Technologies for Large Data Stores*. In Proc. of Scalable Semantic Systems Workshop (SSWS 2007).
- [13] Swiss Institute of Bioinformatics. (2007). *UniProt RDF*. <http://dev.isb-sib.ch/projects/uniprot-rdf/>
- [14] ter Horst, H. J. (2005) 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.