Enabling the business-based
Internet of Things and Services

(FP7 257852)

D4.1 Analysis of Semantic Stores and Specific ebbits Use Cases

Published by the ebbits Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2009.1.3: Internet of Things and Enterprise environments

Document control page

Document file:	D4.1 Analysis of Semantic Stores and Specific ebbits Use Cases.doc
Document version:	2.0
Document owner:	Martin Knechtel (SAP)
Work package:	WP4 – Semantic Knowledge Infrastructure
Task:	T4.1 Enhancement of Semantic Stores
Deliverable type:	R
Document status:	☒ approved by the document owner for internal review ☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Martin Knechtel (SAP)	2011-01-12	Outline and work distribution
0.11	Jan Hreno (TUK)	2011-01-21	Reasoning chapter outline
	Ferry Pramudianto (FIT)	2011-01-27	Section on relation to ebbits Deliverable D5.1.1, Section 5 "Multi-sensor Data Fusion" added to sections 3.4.1 and 4.3
0.12	Matts Ahlsén (CNet)	2011-01-31	Section 3.4 outline
0.13	Martin Knechtel (SAP)	2011-02-08	Sections 3.2, 4.1 on Scalability and queries
0.14	Jan Hreno (TUK)	2011-02-11	Sections 3.3 and 4.2 about reasoning
0.15	Matts Ahlsén (CNet)	2011-02-11	Section 3.4 about Distribution and Centralization
0.16	Martin Knechtel (SAP)	2011-02-11	Conclusions and Executive Summary
0.17	Karol Furdik (IS)	2011-02-11	Contribution to Sections 3.3 and 4.2 about reasoning
1.0	All contributors	2011-02-11	Version for internal review
	Reviewers	2011-02-18	Internal reviews back
2.0	All contributors	2011-02-25	Revisions with respect to reviewer comments
		2011-02-28	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Roberto Checcozzo (COMAU)	2011-02-15	Approved. I added some information on what the semantic store in 2.1. The document is really interesting and I think it is a good job!
Michael Jacobsen (TNM)	2011-02-15	Approved. Various minor corrections. Requests for clearification at places.

Index:

1. Executive summary

This Deliverable forms the starting point for Task 4.1 of the ebbits project. It presents the state of the art of semantic stores and brings it into relation to the ebbits use cases in order to identify the gap to be filled during the ebbits project in Work Package 4 "Semantic Knowledge Infrastructure". In this work package will be considered the following issues related to the available database systems:

- Scalability
- Query complexity
- Distribution
- Architecture

The present document presents the state of the art of semantic stores, providing information related to the aspects of scalability of RDF stores and query performance, through the analysis of experimental data, describing the most promising triple store products that could be used in the ebbits project, and considering the possible centralized and distributed strategies in the development of the storage system.

These three aspects are picked up in Section 4 again when they are brought into relation to the ebbits use cases and requirements.

2. Introduction

2.1 Purpose, context and scope of this deliverable

This Deliverable is a result of Task 4.1 "Enhancement of Semantic Stores" and presents the state of the art of semantic stores and brings it into relation to the ebbits use cases in order to identify the gap to be filled during the ebbits project in Work Package 4 "Semantic Knowledge Infrastructure".

Since Task 4.1 is performed during the complete project time of 4 years, and this Deliverable is due after the first 6 months, it can only be considered as a starting point that helps identifying what can be done in order to advance the state of the art of semantic stores to turn the ebbits vision into reality.

2.2 Background

This Deliverable forms the starting point for Task 4.1 of the ebbits project. The goal of Task 4.1 is the following (citation from the ebbits Description of Work).

The aim of this task is to perform investigation and enhancement of high performance semantic stores using distributed / hierarchical access and manipulation methods. Semantic store provides a basic persistency level for semantic information systems. These systems are not new, and a wide range of supportive tools is available. Some stores available today are Sesame, Jena (which can be used as RDF-store, although it is not a classic RDF-store, RDF-Match (Oracles integration of RDF data, which also makes combined queries with relational data possible), Yars2, 3store, RDF-3X (quite new and it is fast, since it creates indices of all possible permutations), and Hexastore (the approach is similar to RDF-3X). However, there are many open issues to be investigated in order to develop improved semantic stores.
The following points, which are weak points in available systems, will be addressed:

- *Scalability: Most systems have been tested with large scale, but not with very large scale, datasets. It is unclear how they will scale with very large datasets. The task will perform analyses with real data in order to provide a qualified state-of-the-art analysis.*
- *To allow more complex query constructs: The stores usually have problems with multiple RDF triple patterns. The presence of multiple join constructs makes the processing slower. Applicable, new optimizations will be pursued and tested.*
- *Query types: The stores are efficient in queries with subjects. In the ebbits, a similar performance must be achieved with objects and properties in queries.*
- *Reasoning: Often reasoning capabilities of the RDF stores are rather lightweight. It is necessary to allow for more reasoning power while keeping the scalability performance.*
- *Distribution/Centralization: It is unclear how to collect data from the Web to a central repository for processing and efficient querying. Similar to common search engine technology, a semantic index will be designed, which stores knowledge representations found in the Web at a central place. Also search engines have the problem of keeping their index current. In the case of semantic stores, the knowledge base has to be kept current. In scenarios where the Web data often changes, strategies are needed to pull the data adaptively in order to process the freshest possible data.*

The work on scalability and queries will be performed by SAP. TUK and IS will support with reasoning and CNET and FIT will support distribution and centralization.

3. State of the art of semantic stores

3.1 Semantic stores

Classical database management system (DBMS), provide logical data structure that cannot totally satisfy the requirements for a conceptual definition of data, because it is limited in scope and is strongly oriented toward the DBMS software implementation. Therefore, the need to define data from a conceptual view has led to the development of semantic data modeling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data.

A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world, enhancing the meaning of the data. In the ebbits project, semantic stores provide a significant help in the simplification of the database system usage and the matching of the database with the real world.

In this work has been considered the SPARQL language. SPARQL (pronounced "sparkle") is an RDF query language; its name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language. It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is considered a key semantic web technology. On 15 January 2008, SPARQL became an official W3C recommendation, implementations for multiple programming languages exist.

SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

3.2 Scalability and queries

This section describes the scalability of RDF (Resource Description Framework) stores for large datasets and the performance of SPARQL queries.

Specifics of SPARQL queries have been investigated in the PhD thesis (Schmidt 2010). The thesis includes

- a complete complexity analysis for all operator fragments of the SPARQL query language, which identifies operator constellations that make query evaluation hard and - as a central result - shows that the SPARQL OPTIONAL operator alone, which allows for the optional selection of components in RDF graphs, is responsible for the PSpace-completeness of the SPARQL evaluation problem;
- a language-specific benchmark suite for SPARQL, called SP2Bench, which allows to assess the performance of SPARQL implementations in a comprehensive, application-independent setting.

The benchmark SP2Bench is not developed around a selected use case but instead highly SPARQL-specific. It covers a variety of challenges that engines may face when processing RDF(S) data with SPARQL. The data generator is complemented by a set of 17 benchmark queries, specifically designed to test characteristic SPARQL operator constellations and RDF access patterns over the generated documents. The thesis (Schmidt 2010) allows insights for SPARQL endpoint implementers on the complexity of SPARQL evaluation and provides an algebraic SPARQL query optimization. Once the implementation is ready, with SP2Bench it also provides a benchmark to assess the query optimizations. A selected set of available RDF stores has been tested with the SP2Bench benchmark, e.g., in (Schmidt 2009). Those experimental results witnessed that SPARQL implementations like ARQ, Sesame, or Virtuoso suffered from severe performance bottlenecks when dealing with medium- and large-scale RDF databases, even for presumably simple queries that can be processed efficiently in a comparable relational setting.

The W3C publishes a collection of RDF store benchmarks at (W3C 2010). The collection consists of academic publications, test sets, benchmarks and benchmarking results. The benchmarks contain the

- Berlin SPARQL Benchmark (BSBM), which provides a comparison of the performance of RDF and Named Graph stores as well as RDF-mapped relational databases and other systems that expose SPARQL endpoints,
- Lehigh University Benchmark (LUBM), which evaluates the performance of repositories with respect to extensional queries over a large data set,
- Ontology Benchmark (UOBM) extends the LUBM benchmark in terms of inference and scalability testing,
- JustBench, which analyses the performance of OWL reasoners based on justifications for entailments.

Since the results from LUBM in (Guo et al. 2005) and UOBM in (Ma et al. 2006) can be considered quite old at the time of this Deliverable, we focus on the results of BSBM and JustBench in further detail.

3.2.1 Results from the Berlin SPARQL Benchmark (BSBM)

The results from the BSBM are published in (Bizer and Schultz 2009). The benchmark is introduced as a "benchmark for comparing the performance of storage systems that expose SPARQL endpoints. Such systems include native RDF stores, Named Graph stores, systems that map relational databases into RDF, and SPARQL wrappers around other kinds of data sources" (Bizer and Schultz 2009).

The test data is "built around an e-commerce use case, where a set of products is offered by different vendors and consumers have posted reviews about products." The datasets consist of 1,000,000 to 100,000,000 triples and can be downloaded at (Bizer and Schultz 2009). The datasets already contain all inferences, so that the systems under test did not have to do any inferencing.

Results from the BSBM are available for
- four RDF stores (Virtuoso Version 5.0.10, Sesame Version 2.2.4, Jena TDB Version 0.72, Jena SDB Version 1.2.0) and
- two relational database-to-RDF wrappers (D2R Server Version 0.6 and Virtuoso - RDF Views Version 5.0.10).
- two SQL versions of the benchmark in order to bring the SPARQL results into context to relational database management systems (MySQL 5.1.26 and Virtuoso - RDBMS Version 5.0.10)

The benchmarks have been run on a Linux PC with Intel Core 2 Quad Q9450 2.66GHz, 8GB RAM, 160GB HDD with 10,000 rpm and 750GB HDD with 7,200 rpm. The exact hardware configuration and test procedure is given at (Bizer and Schultz 2009).

The test driver and the system under test (SUT) were running on the same machine in order to reduce the influence of network latency. The test driver issues the commands to load the datasets and to query the datasets.

Table 1 provides the results about the load times required by the SUTs.

Table 1: Load times for SUTs and the different datasets in [day:]hh:min:sec (Bizer and Schultz 2009)

SUT	1M	25M	100M
Sesame	00:02:59	12:17:05	3:06:27:35
Jena TDB	00:00:49	00:16:53	01:34:14
Jena SDB	00:02:09	04:04:38	1:14:53:08
Virtuoso TS	00:00:23	00:39:24	07:56:47
Virtuoso RV	00:00:34	00:17:15	01:03:53
D2R Server	00:00:06	00:02:03	00:11:45
MySQL	00:00:06	00:02:03	00:11:45
Virtuoso SQL	00:00:34	00:17:15	01:03:53

The query performance of the SUTs has been measured with different mixes of queries, containing altogether 12,500 queries. The queries were issued against the system with the SPARQL protocol (with the SQL protocol for the RDBMS comparison). The results in Table 2 compare the SPARQL query performance of the different stores and put them into relation to the SQL query performance of MySQL and Virtuoso's SQL engine. The SQL performance figures also allow calculating the overhead that is produced by the relational database to RDF wrappers when rewriting SPARQL queries into SQL queries against the underlying RDBMS. The two SQL systems are no "SQL backed semantic stores" but pure SQL systems. The SPARQL queries have been manually rewritten to SQL queries. The measured performance is very good, but as explained, both are no semantic store. For this reason the measured performance cannot be taken into account when pointing out the fastest semantic store. (Bizer and Schultz 2009).

Table 2: Query mixes per hour. Best performance in dataset (excluding SQL engines) is bold (Bizer and Schultz 2009)

	Sesame Native	Jena TDB	Jena SDB	Virtuoso TS	Virtuoso RV	D2R Server	MySQL SQL	Virtuoso SQL
1 M	**18,094**	4,450	10,421	12,360	17,424	2,828	*235,066*	*192,013*
25 M	1,343	353	968	4,123	**12,972**	140	*18,578*	*69,585*
100 M	254	81	211	954	**4,407**	35	*4,991*	*9,102*

The following Tables provide results about performance in answering queries from a single client. Each of them contains one line per query type. The details about the used query types can be found in (Bizer and Schultz 2009). Table 3 presents the results with a 1M triple dataset, Table 4 with a 25M triple dataset and Table 5 with a 100M triple dataset. While for smaller datasets Sesame outperforms the other stores in almost all cases, for bigger datasets especially Virtuoso RV shows a more performant behavior.

Table 3: Queries per second. 1M triple dataset. Best performance in dataset (excluding SQL engines) is bold (Bizer and Schultz 2009)

	Sesame Native	Jena TDB	Jena SDB	Virtuoso TS	Virtuoso RV	D2R Server	MySQL SQL	Virtuoso SQL
Query 1	**662**	494	374	202	199	328	*3,021*	*1,195*
Query 2	**251**	61	50	47	78	41	*4,525*	*1,592*
Query 3	**505**	451	283	176	182	226	*2,833*	*1,079*
Query 4	**452**	429	240	92	106	224	*2,653*	*1,098*
Query 5	30	2	18	76	**118**	1	*396*	*411*
Query 6	14	60	17	55	**275**	26	*164*	*1,605*
Query 7	87	**189**	112	72	81	123	*1,912*	*831*
Query 8	**297**	159	134	116	132	72	*3,497*	*1,715*
Query 9	**924**	57	129	541	506	81	*4,255*	*2,639*
Query 10	**429**	**429**	289	95	224	218	*4,444*	*2,004*
Query 11	**652**	376	351	361	102	33	*9,174*	*2,494*
Query 12	**797**	53	119	133	151	203	*7,246*	*2,801*

Table 4: Queries per second. 25M triple dataset. Best performance in dataset (excluding SQL engines) is bold (Bizer and Schultz 2009)

	Sesame Native	Jena TDB	Jena SDB	Virtuoso TS	Virtuoso RV	D2R Server	MySQL SQL	Virtuoso SQL
Query 1	200	165	198	192	173	**236**	*955*	*833*
Query 2	**168**	51	47	46	75	36	*3,333*	*1,456*
Query 3	140	141	151	165	**167**	115	*919*	*838*
Query 4	128	116	132	86	96	**167**	*919*	*759*
Query 5	2	0.1	1	14	**30**	0.04	*25*	*43*
Query 6	1	2	1	2	**25**	1	*7*	*97*
Query 7	57	28	27	36	76	**97**	*1,370*	*733*
Query 8	90	27	30	113	**129**	62	*601*	*1,603*
Query 9	128	3	9	**533**	482	73	*2,849*	*2,639*
Query 10	93	62	40	75	**220**	200	*3,356*	*1,587*
Query 11	98	45	97	**342**	100	2	*4,367*	*3,195*
Query 12	**350**	3	9	129	148	162	*2,571*	*2,985*

Table 5: Queries per second. 100M triple dataset. Best performance in dataset (excluding SQL engines) is bold (Bizer and Schultz 2009)

	Sesame Native	Jena TDB	Jena SDB	Virtuoso TS	Virtuoso RV	D2R Server	MySQL SQL	Virtuoso SQL
Query 1	15	35	12	**132**	122	79	*476*	*470*
Query 2	32	38	35	39	**64**	40	*3,268*	*991*
Query 3	13	28	8	**136**	129	56	*459*	*456*
Query 4	10	25	7	54	**84**	72	*428*	*443*
Query 5	0.5	0.04	0.5	5.9	**13.6**	0.01	*7.9*	*12.2*
Query 6	0.1	0.1	0.1	0.5	**6**	0.2	*1.9*	*21.7*
Query 7	2	6	2	5	**15**	12	*407*	*26*
Query 8	4	8	3	12	**22**	12	*63*	*31*
Query 9	19	1	2	53	**164**	33	*1,370*	*145*
Query 10	2	19	2	8	67	**77**	*1,883*	*267*
Query 11	13	24	23	**44**	41	0	*456*	*1,248*
Query 12	18	1	2	39	91	**170**	*539*	*1,524*

The following Tables provide results about performance in answering queries from multiple clients. In real world scenarios there are usually more than one clients working against a SPARQL endpoint. For this reason the following results are more relevant than the ones above for one client. Table 6 and Table 7 provide the results. The number of query mixes per hour has been extrapolated from the time it took all clients together to execute 500 query mixes. For the detailed test procedure we refer the reader to (Bizer and Schultz 2009). Again, Virtuoso shows the best performance, followed by Sesame. As explained above, the two pure SQL systems are not taken into account for this comparison since they are no semantic store.

Table 6: Query mixes per hour. 1M triple dataset. Best performance in a fixed number of client (excluding SQL engines) is bold (Bizer and Schultz 2009)

Dataset Size 1M	Number of clients				
	1	2	4	8	64
Sesame	**18,094**	19,057	16,460	18,295	16,517
Jena TDB	4,450	6,752	9,429	8,453	8,664
Jena SDB	10,421	17,280	23,433	24,959	23,478
Virtuoso TS	12,360	21,356	32,513	29,448	29,483
Virtuoso RV	17,424	**28,985**	**34,836**	**32,668**	**33,339**
D2R Server	2,828	3,861	3,140	2,960	2,938
MySQL	*235,066*	*318,071*	*472,502*	*442,282*	*454,563*
Virtuoso SQL	*192,013*	*199,205*	*274,796*	*357,316*	*306,172*

Table 7: Query mixes per hour. 25M triple dataset. Best performance in a fixed number of client (excluding SQL engines) is bold (Bizer and Schultz 2009)

Dataset Size 25M	Number of clients				
	1	2	4	8	64
Sesame	1,343	1,485	1,204	1,300	1,271
Jena TDB	353	513	694	536	555
Jena SDB	968	1,346	1,021	883	927
Virtuoso TS	4,123	7,610	9,491	5,901	5,400
Virtuoso RV	**12,972**	**22,552**	**30,387**	**28,261**	**28,748**
D2R Server	140	187	160	146	143
MySQL	*18,578*	*31,093*	*39,647*	*40,599*	*40,470*
Virtuoso SQL	*69,585*	*85,146*	*135,097*	*173,665*	*148,813*

3.2.2 Results from the JustBench Benchmark

Since OWL reasoners can be connected to state of the art RDF stores to make implicit knowledge explicit, we review an approach for benchmarking OWL reasoners. The performance analysis of OWL reasoners on expressive OWL ontologies is an ongoing challenge. In contrast to other reasoning benchmarks, JustBench does not measure the time for a full computation of all consequences but it allows for a more fine granular assessment. The assessment can be done per consequence. This helps to identify the type of consequences which causes performance problems to a given reasoner.

The JustBench setting and results have been published in (Bail, Parsia and Sattler 2010). The test system is a Mac Pro desktop system (2.66 GHz Dual-Core Intel Xeon processor, 16 GB physical RAM) with 2GB of memory allocated to the Java virtual machine. The three reasoners FaCT++ 1.4.0, HermiT 1.2.3, and Pellet 2.0.1 have been tested. The set of test ontologies consists of Building, Chemical, Not-Galen (a modified version of the Galen ontology), DOLCE Lite, Wine and MiniTambis[1]

As a preparation step, for each consequence of every ontology, the set of justifications has been extracted. A justification is a minimal set of axioms that entails a consequence. The number of justifications for each entailment ranged from 1 to over 300, with the largest containing 36 axioms.

In the test series, the time was measured that a reasoner needed in order to decide whether a consequence follows from a justification. The reasoner has to decide with “yes” if it works correctly. This was not always the case and details on the observed incorrectness of Pellet are given in the paper. It should be noted, that only consequences which should have been but haven’t been computed can be detected but not the inverse. It is not possible to detect non-entailments. Thus, the test is not analytically complete, but the authors claim that it still scores high on understandability. The incorrectness has been submitted to the developers and might have been corrected meanwhile.

The interesting parameter in the context of this Deliverable is the time required. The time required is given in Figure 1 for the MiniTambis ontology and in Figure 2 jointly for all ontologies. Especially the performance of the reasoner Pellet decreases with higher justification sizes. FaCT++ is the fastest of all 3 reasoners.

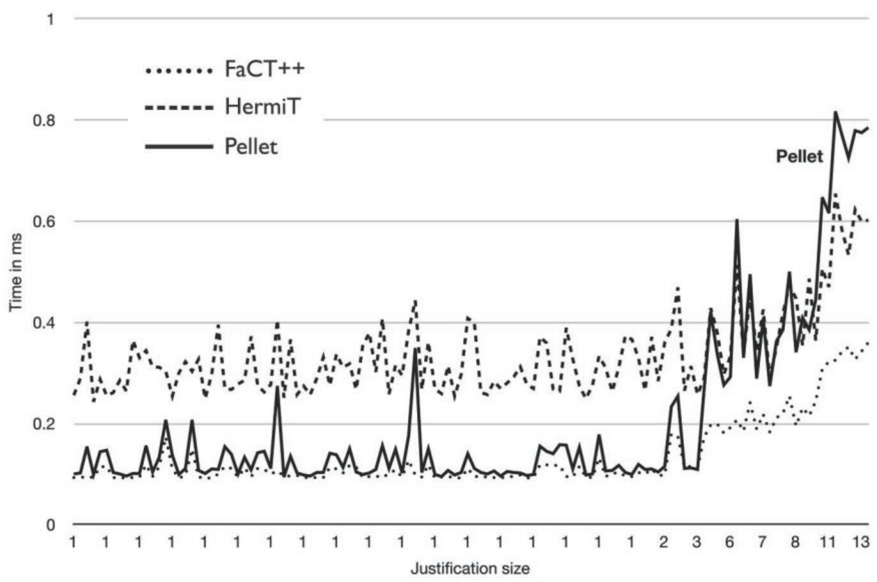

Figure 1: Performance of reasoners with the MiniTambis ontology depending on the size of justifications (Bail, Parsia and Sattler 2010)

[1] The ontologies can be found online at http://owl.cs.man.ac.uk/explanation/justbenchmarks

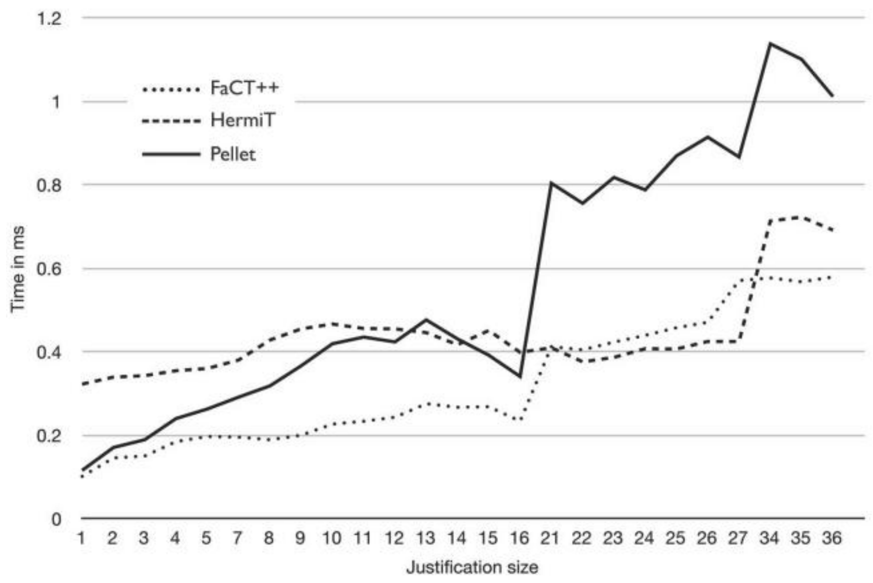

Figure 2: Performance of reasoners depending on the size of justifications (Bail, Parsia and Sattler 2010)

A second experiment involves an artificially generated ontology. The results are given in Figure 3. It turns out that entailments of this ontology have significantly more justifications. While HermiT takes comparably long for loading the ontology, it is comparably fast for checking if a justification entails a consequence.

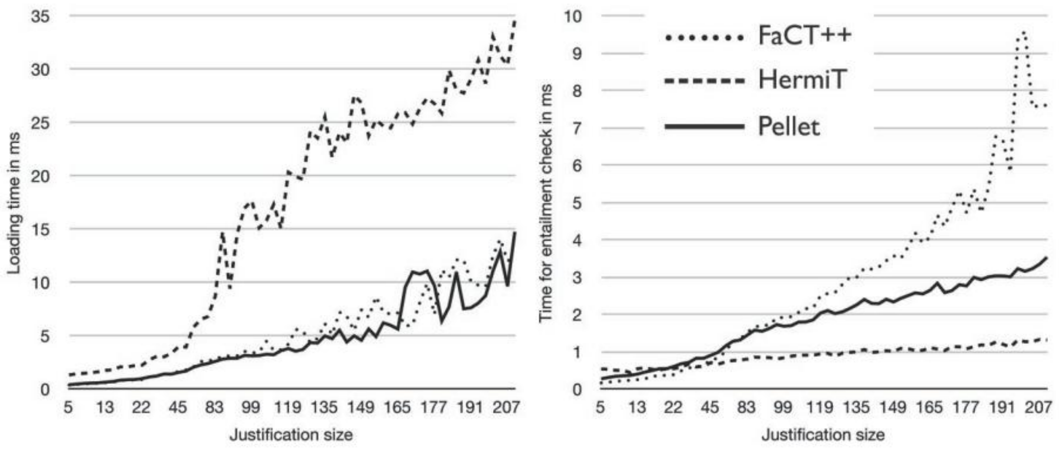

Figure 3: Reasoner performance on an artificially generated ontology (Bail, Parsia and Sattler 2010)

3.3 Reasoning

The reasoning can be defined as an ability to infer a qualitatively new information from a set of asserted facts or axioms stored in a semantic repository. It means that the reasoning goes behind a simple retrieval or querying of ontologies to obtain the explicitly stored facts, i.e. to find all the triplets that contain particular object, predicate, etc. The task of reasoning is somehow broader; it is focused on a retrieval of facts that are implicit in the ontology and can be derived by means of a combination of explicitly stated facts (Davies et al. 2006), (Serrano et al. 2007).

The process of reasoning is based on an identification of logical consequences that can be extracted from the semantically enriched data. Particular solutions depend on a representation of underlying semantic structures that determine the implementation of reasoners. The reasoning mechanisms typically employ the first-order predicate logic or other type of description logic, where rules are applied for forward and backward chaining (Baader and Nutt 2002). Another group of reasoners, which can be seen as an alternative to the rule-based systems, is built on statistical or probabilistic approaches (Pearl 1997).

Assuming that ontologies can be seen as simplified models of human memory, which are capable to store and represent knowledge of a given domain, the reasoning applied on such a formal semantic structure can be related to the human consciousness. Namely, the induction and deduction are two basic types of logical inference, which correspond to the forward and backward chaining procedures, respectively. The relations of generalisation (e.g. is_a, part_of, has, etc.) are utilised for extracting a general goal from atomic facts given on an input (forward chaining, inductive inference) or for determining atomic facts that may satisfy an input general goal (backward chaining, deductive inference). Besides the extraction of logically inferred information, reasoners are employed for ontology merging and solving inconsistency problems between possibly heterogeneous networked ontologies (Davies et al. 2006).

The expressiveness and performance of reasoners depends on particular description logic variant and the respective algorithm implemented in the reasoning engine of a semantic store. Language variants refer to RDFS fragments of description logic or OWL sub-languages such as OWL Lite / DL / Full or OWL 2 profiles as EL / QL / RL (Baader 2009). Benchmarks of existing reasoners vary in evaluation methods; however, the criteria of query selectivity, complexity, subsumption, response and elapsed time are typically included (Lee et al. 2008), (Bock et al. 2008). There exists a relatively wide range of available reasoners, some of the most commonly used are referenced by the W3C OWL working group (W3C Implementations 2010). However, for the purposes of ebbits, we will focus the survey and analysis of suitable reasoners to the solutions that integrate reasoning engines of high performance into a comprehensive semantic repository platform.

3.3.1 Survey

As we consider working with a huge amount of sensor data in the ebbits cases, which has to be potentially stored, accessed and reasoned about in a triple store, we have identified some of already available triple stores that can fulfil these expectations. We will analyse, how these selected triple stores deal with a huge amount of data, and how these products can be used in the heterogeneous distributed environment of the ebbits use cases. We will focus on analysis of these properties of the products:

- Formalisms supported
- API implementation
  - We will focus here on Java, as a technology of choice for the ontology manager of ebbits.
- Reasoner engine implementation
- Federation support
- Datatype support
- Full text search support
- Native extensions

Some of the most known triple stores (shortly described in a technology watch report D2.2.1) were surveyed:

- BigOWLIM, SwiftOWLIM[2]
- Bigdata[3]
- AllegroGraph[4]
- OntoBroker[5]
- Sesame[6]

[2] http://www.ontotext.com/owlim/
[3] http://www.bigdata.com/bigdata/blog/
[4] http://www.franz.com/agraph/allegrograph/
[5] http://www.ontoprise.de/en/home/products/ontobroker/

- Jena[7]

According to the survey, we can divide triple stores into 2 basic groups. Triple stores using relational databases for storing data and triple stores using their own (native) storage mechanisms. We have selected one representative from both groups to further investigate their capabilities. Two of best performing and most scalable solutions were selected (based on publicly available benchmarks). The selected triple stores are BigOWLIM (capable of efficiently working with up to 20B triples, LUBM 8000 load rate 20,6K/sec with OWL-Horst inference activated) and AllegroGraph (20+B triples, LUBM 8000 load rate 303K/sec with 6 indices activated).

3.3.2 AllegroGraph 4.2

Overview

AllegroGraph[8] is a database and application framework for building Semantic Web applications.

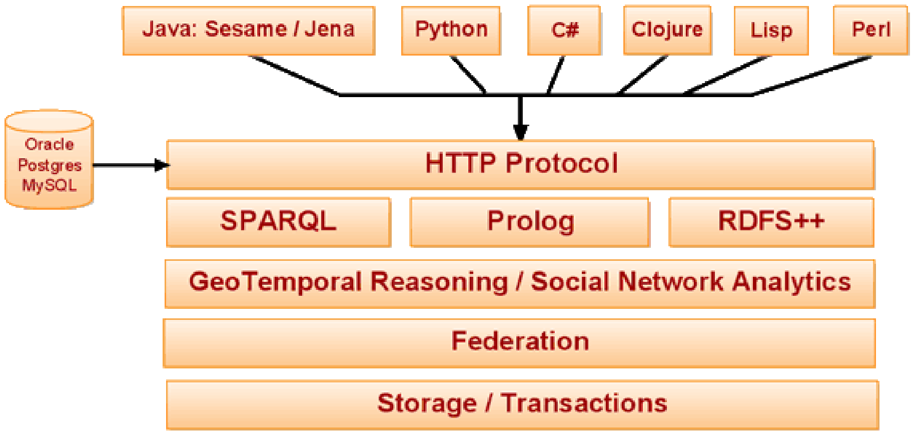

Figure 4: The AllegroGraph Architecture (Source: http://www.franz.com)

It can store data and meta-data as triples; query these triples through various query APIs like SPARQL and Prolog; and apply RDFS++ reasoning with its built-in reasoner. AllegroGraph includes support for Federation, Social Network Analysis, Geospatial capabilities and Temporal reasoning. The basic architecture of the AllegroGraph framework is given in Figure 4.

Licence

AllegroGraph RDF Store is available in three editions:

- Free for < 50 Million Triples
- Developer for < 600 Million Triples
- Enterprise for Unlimited Triples

System Requirements

The AllegroGraph version 4 Server runs natively on Linux x86-64 bit. To run AllegroGraph version 4 on other operating systems (i.e. Windows, Mac) Virtual Machine images are provided. Clients to an AllegroGraph server may be either 32-bit or 64-bit.

The developers promised, that native implementations for Apple Mac OSX (x86-64) 10.6 and 64-bit Microsoft Windows 2000/XP/Vista/7/Server 2003 are coming soon.

[6] http://www.openrdf.org/
[7] http://jena.sourceforge.net/
[8] http://www.franz.com/agraph/allegrograph/

For native 64-bit Mac, Windows, and Solaris, and all 32-bit systems, one can also use AllegroGraph 3.3 version.

Implementation

The AllegroGraph triple-store is composed of assertions. Though called triples, each assertion has five fields:

- subject (s)
- predicate (p)
- object (o)
- graph (g)
- triple-id (i)

All of s, p, o, and g are strings of arbitrary size. To speed queries, AllegroGraph creates indices which contain the assertions plus additional information. AllegroGraph can also perform freetext searching in the assertions using its freetext indices. Finally, AllegroGraph keeps track of deleted triples. Advantages over Relational Database Management Systems (RDBMSs) are as follows:

- New predicates can be added without changing any schema
- One-to-many relations are directly encoded without the indirection of tables
- Everything is automatically indexed

AllegroGraph implements the ACID properties of transaction processing (atomicity, consistency, isolation, and durability) similar to other database products.
The atomicity property defines that all updates within one transaction are persisted together. A transaction either completely fails or completely succeeds. The consistency property defines that every transaction takes the database as a whole from one consistent state to another. The database itself will never be inconsistent, according to its own consistency rules. AllegroGraph does not allow for user-defined consistency rules (like, say, foreign key constraints in a relational database). It is up to the user to make sure that transactions create and maintain a consistent application-level state. The isolation property defines that every transaction only sees data of other completed transactions, and not partial results of transactions running concurrently. As AllegroGraph performs no triple locking, it is possible that a triple that is being read in a transaction could be deleted in a concurrent transaction. Developers need to be aware of this and similar possibilities and make sure that transactions are properly sequenced if such concurrent updates could have an impact on application-level consistency. The durability property defines that once the database system signals the successful completion of a transaction to the application, the changes made by the transaction will persist even in the presence of hardware and software failures.

Adding triples

AllegroGraph supports N-Triples, RDF/XML, Turtle files for import. Parsers for N3 and other common file formats are planned for the future. One can also load triples into AllegroGraph programmatically. This can be used to import custom data formats, or to build a triple-store incrementally. Triples can be added using RDF syntax or AllegroGraph's special encoded data-types.

Federation

AllegroGraph uses that same programming API to connect to local triple-stores (either on-disk or in-memory), remote-triple-stores and federated triple-stores. A federated store collects multiple triple-stores of any kind into a single virtual store that can be manipulated as if it were a simple local-store. Federation provides three big benefits:

- scalability
    - use separate instances of AllegroGraph to load data on multiple CPUs
- manageability

    - keeping known facts, inferred triples, provenance information, ontologies, metadata and deleted triples in separate, easily manageable stores and combine and re-combine the data as necessary
- easy archiving
    - federation allows segmenting of data into usable chunks that can be swapped in and out as needed

Querying

AllegroGraph gives several options for extracting data from RDF graphs. Most basic is the API itself, working with individual triples. A logic view is offered by Prolog. SPARQL more closely resembles SQL, and offers a relational, pattern-based approach to retrieving data from a store. AllegroGraph's SPARQL implementation is called twinql
Conceptually, twinql has three layers:

- a parser from the textual SPARQL surface;
- a query builder and planner that prepares the query for execution;
- and an executor that runs the plan against a store to produce results.

Currently, input and output from each of these layers is limited (for example, the query plan is not available to user code, but parsed output is). This will change in a future release.

There are three possible outputs from a SPARQL query:

- a yes/no answer, in response to an ASK query;
- a list of bindings, in response to a SELECT query; or
- a new RDF graph, in response to a CONSTRUCT or DESCRIBE query.

AllegroGraph Triple Indices
AllegroGraph uses a set of sorted indices to quickly identify a contiguous block of triples that are likely to match a specific query pattern. These indices are identified by names that describe their organization. The default set of indices are called spogi, posgi, ospgi, gspoi, gposi, gospi, and i, where:

- s stands for the subject URI.
- p stands for the predicate URI.
- o stands for the object URI or literal.
- g stands for the graph URI.
- i stands for the triple identifier (its unique id number within the triple store).

Reasoning

AllegroGraph's RDFS++ reasoning supports all the RDFS predicates and some of OWL's. It is not complete but it has predictable and fast performance. Here are the supported predicates:

- rdf:type and rdfs:subClassOf
- rdfs:range and rdfs:domain
- rdfs:subPropertyOf
- owl:sameAs
- owl:inverseOf
- owl:TransitiveProperty

In addition to RDFS++ reasoning, AllegroGraph also supports reasoning over hasValue restrictions in equivalent classes or subclasses.

- owl:hasValue
- owl:someValuesFrom
- owl:allValuesFrom

Prolog

Prolog is an alternative query mechanism for AllegroGraph. With Prolog, one can specify queries declaratively.

Datatypes and extensions

AllegroGraph supports several datatypes for efficient storage, manipulation, and search of strings, numbers, dates, Social Network data, Geospatial and Temporal information.

Basic data-types

AllegroGraph stores a wide range of data types directly in its low level triple representation. This allows for very efficient range queries and significant reduction in triple-store data size. With other triple-stores that only store strings, the only way to do a range query is to go through all the values for a particular predicate. This works well if everything fits in memory; but if the predicate works with millions of triples, it will need costly machines with huge amounts of RAM. AllegroGraph supports most XML Schema types (native numeric types, dates, times, longitudes, latitudes, durations and telephone numbers).

Social Network Analysis

By viewing interactions as connections in a graph, we can treat a multitude of different situations using the tools of Social Network Analysis (SNA). SNA lets us answer questions like:

- How closely connected are any two individuals?
- What are the core groups or clusters within the data?
- How important is this person (or company) to the flow of information
- How likely is it that this person and that person know one another

AllegroGraph's SNA toolkit includes an array of search methods, tools for measuring centrality and importance, and the building blocks for creating more specialized measures.

Geospatial Primitives

AllegroGraph provides a novel mechanism for efficient storage and retrieval of geospatial data. 3 Support is provided both for Cartesian coordinate systems (i.e., a flat plane) and for spherical coordinate systems (e.g., the surface of the earth or the celestial sphere). AllegroGraph's geospatial application also has support for defining polygons and quickly determining position according these polygons.

Temporal Primitives

AllegroGraph supports efficient storage and retrieval of temporal data including datetimes, time points, and time intervals:

- datetimes in ISO8601 format: "2008-02-01T00:00:00-08:00"
- time points: ex:point1, ex:h-hour, ex:when-the-meeting-began, etc
- time intervals: ex:delay-interval (say, from point ex:point1 to ex:h-hour)

Once data has been encoded, applications can perform queries involving a broad range of temporal constraints on data, including relations between :

- points and datetimes
- intervals and datetimes
- two points
- two intervals
- points and intervals

Freetext Indexing

AllegroGraph can build freetext indexes of the strings of the objects associated with a set of predicates that one specify. Given a freetext index, one can search for text using:

- boolean expressions ("market" AND "housing")
- wild cards ("science*" OR "math*")
- phrases ("Semantic Web search")

Freetext indexing slows the rate at which one can insert triples between 5 and 25% depending on the number of predicates involved and the kinds of string data in the application.

AllegroGraph supports multiple free-text indices, each targeted on specific fields of specific predicates. These text indices are based on a locality-optimized Radix tree for intelligent traversal for fast wildcard and fuzzy searches. The indexing process is fully transactional, and is able to easily handle billions of documents.

Programming with AllegroGraph

AllegroGraph comes in multiple flavors and works with multiple programming languages and environments.

Java

The Java client interface implements most of the Sesame and Jena interfaces for accessing remote RDF repositories. Because AllegroGraph provides functionality not found in other triple-stores, extensions were implemented where applicable. The Java API supports the following operations:

- Creating a Repository and Triple Indices
- Asserting and Retracting Triples
- Statement Matching for simple retrieval
- Work with Literal Values (Numeric, String, Boolean, Date, Time, Datetime)
- Importing Triples (RDF/XML, NTriples)
- Exporting Triples (RDF/XML, NTriples)
- Searching Multiple Graphs
- Namespaces
- Free Text Search
- Execute SPARQL Query
  - Select, Ask, Describe, and Construct Queries
  - Parametric Queries
  - Range Matches
- Federated Repositories
  - AllegroGraph lets one split up triples among repositories on multiple servers and then search them all in parallel. From the point of view of a Java code, it looks like one is working with a single repository.
- Prolog Rule Queries
- RDFS++ Inference
  - AllegroGraph's inference engine can be turned on or off each time one runs a query against the triple store. (Inference is turned off by default, which is the opposite of standard Sesame behavior.)
- Geospatial Search
- Social Network Analysis
- Transactions
  - "Commit" means to make a batch of newly-loaded triples visible in the auto-commit connection. The two sessions are "synched up" by the commit. Any "new" triples added to either connection will suddenly be visible in both connections after a commit.
  - "Rollback" means to discard the recent additions to the transaction connection. This, too, synchs up the two sessions. After a rollback, the transaction connection "sees" exactly the same triples as the auto-commit connection does.
  - "Closing" the transaction connection deletes all uncommitted triples, and all rules, generators and matrices that were created in that connection. Rules, generators and matrices cannot be committed.
- Eliminating Duplicate Triples
  - Filter Out Duplicate Results
  - Filtering Duplicate Triples while Loading
  - Working with Duplicates in Federated Stores

HTTP

It is possible for web developers and programmers alike to interact with AllegroGraph 4.2 completely using a RESTful HTTP protocol (using GET, PUT, POST) to add and delete triples, to query for individual triples and to do SPARQL and Prolog selects using the Sesame 2.0 HTTP-interface with some extensions

Python

The Python API offers convenient and efficient access to an AllegroGraph server from a Python-based application. This API provides methods for creating, querying and maintaining RDF data, and for managing the stored triples.

Lisp

Lisp programmers can open and use triple-stores from within Lisp. Lispers can create applications in the same image that the AllegroGraph server is running or use a remote-triple-store to access data in client/server mode.

Other clients based on http REST Protocol include C#, Clojure, Perl, Ruby, Scala clients.

AllegroGraph is compatible with other semantic technologies or products. These include TopBraid Composer[9], RacerPro[10] reasoning system, AGWebview[11] web based managing system for AllegroGraph, Gruff[12] triple-store visual browser, Pepito[13] data mining system, Cogito[14] search extraction and classification system, Sentient Suite[15] knowledge and project management system.

3.3.3 OWLIM

Overview

OWLIM[16] is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM is based on Ontotexts's Triple Reasoning and Rule Entailment Engine (TRREE). The two editions of OWLIM are SwiftOWLIM and BigOWLIM. In SwiftOWLIM, reasoning and query evaluation are performed in-memory, while, at the same time, a reliable persistence strategy assures data preservation, consistency, and integrity. BigOWLIM is the high-performance 'enterprise' edition that scales to massive quantities of data. Typically, SwiftOWLIM can manage millions of explicit statements on desktop hardware, whereas BigOWLIM can manage billions of statements and multiple simultaneous user sessions.
of query languages (e.g. SPARQL and SeRQL) and RDF syntaxes (e.g. RDF/XML, N3, Turtle).

Licence

Downloading and use of SwiftOWLIM is free of charge for any purpose. BigOWLIM is provided free of charge for research, evaluation and development purposes. Ontotext offer maintenance packages and commercial licences for BigOWLIM.

System requirements

OWLIM can be installed on Java JRE version 1.5 onwards (both 32-bit and 64-bit versions). If custom rule-sets are used then a Java JDK version 1.6 inwards is required.

[9] http://www.franz.com/agraph/tbc/
[10] http://www.franz.com/agraph/racer/
[11] http://www.franz.com/agraph/agwebview/
[12] http://www.franz.com/agraph/gruff/
[13] http://www.franz.com/products/pepito/
[14] http://www.expertsystem.net/page.asp?id=1515&idd=200&lang=1
[15] http://www.io-informatics.com/
[16] http://www.ontotext.com/owlim/

Implementation

SwiftOWLIM and BigOWLIM are identical in terms of usage and integration. The editions differ in the respective version of the TRREE engine they are based upon, but share the same inference mechanisms and semantics (rule-compiler, etc). The different versions of the TRREE engine use different indexing, inference and query evaluation implementations, which results in different performance, memory requirements, and scalability.

SwiftOWLIM is a fast semantic repository. Its key features are:

- reasoning and query evaluation performed in main memory;
- persistence strategy that assures data preservation and consistency;
- extremely fast loading of data (including inference and storage).

Although the reasoning is handled in-memory, the SwiftOWLIM SAIL offers a relatively comprehensive persistence and backup strategy. The persistence of SwiftOWLIM is implemented via writing to file in N-Triple format. The repository can be split into several files, where all of these except one are read-only; the writable file is considered as both the source from which the triples are loaded and the target where the new statements are stored. This backup strategy ensures that no loss of newly asserted triples can occur in cases of power failure or abnormal termination.

BigOWLIM is the most scalable semantic repository in the World. The key features of BigOWLIM are as follows:

- The most scalable semantic repository in the World, both in terms of the volume of RDF data it can store and the speed with which it can load and do inferencing;
- Pure Java implementation, ensuring ease of deployment and portability;
- Compatible with Sesame 2, which brings interoperability benefits and support for all major RDF syntaxes and query languages;
- Customisable reasoning, in addition to RDFS, OWL-Horst, and OWL 2 RL support;
- Optimized owl:sameAs handling, which delivers dramatic improvements in performance and usability when huge volumes of data from multiple sources are integrated.
- Clustering support brings resilience, failover and scalable parallel query processing;
- Geo-spatial extensions;
- Full-text search support;
- High performance retraction of statements and their inferences – so inference materialisation speeds up retrieval, but without delete performance degradation;
- Powerful and expressive consistency/integrity constraint checking mechanisms;
- RDF rank;
- RDF Priming, based upon activation spreading, allows efficient data selection and context-aware query answering for handling huge datasets;
- Notification mechanism, to allow clients to react to statements in the update stream.

BigOWLIM supports the so called 'read committed' transaction isolation level, well known to relational database management systems. It guarantees that changes will not impact query evaluation, before the entire transaction they are part of is successfully committed. It does not guarantee that execution of a single transaction is performed against a single state of the data in the repository.

Regarding concurrency:

- multiple update/modification/write transactions can be initiated and stay open simultaneously, i.e. one transaction does not need to be committed in order to allow another transaction to complete;
- update transactions are processed internally in sequence, i.e. OWLIM processes the commits one after another;
- update transactions do not block read requests in any way, i.e. hundreds of SPARQL queries can be evaluated in parallel (the processing is properly multi-threaded) while update transactions are being handled on separate threads.

The limitations of OWLIM are related to its reasoning strategy. In general, the expressivity of the language supported cannot be extended in the Description Logic direction, because the semantics must be able to be captured in (Horn) rules. The total materialisation strategy has drawbacks when changes to the explicitly asserted statements occur frequently. For expressive semantics and certain ontologies, the number of implicit statements can grow quickly with the expected degradation in performance. BigOWLIM has a number of optimisations to reduce this problem, e.g. special handling of owl:sameAs. Removing explicit statements can adversely affect performance if the full closure needs to be recomputed.

Adding triples

The import and export of all major RDF syntaxes (XML, N3, N-Triples, Turtle, TRIG, TRIX) is supported through Sesame.

Federation

The BigOWLIM software suite includes an additional Replication Cluster component that serves as a Master node for a cluster. Its purpose is to manage and distribute atomic requests (query evaluations and update transactions) to a set of standard BigOWLIM instances. The Master node of the BigOWLIM Replication Cluster implements the Sesame Repository interfaces. However, it does not store any RDF data itself, rather its function is to route queries and update requests to a set of standard BigOWLIM instances (nodes). Worker nodes are standard BigOWLIM repositories configured with identical rule-sets hosted in the openrdf-sesame Web application running in a Java servlet container, such as Tomcat. These are accessible by the Master node via the HTTP protocol of the exported SPARQL endpoint of the Sesame service.

Querying

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

Reasoning

The supported semantics can be configured through the definition of rule-sets. The most expressive pre-defined rule-set combines unconstrained RDFS and OWL-Lite. Custom rule-sets allow tuning for optimal performance and expressivity. OWLIM supports RDFS, OWL DLP, OWL Horst, most of OWL Lite and OWL2 RL.

OWLIM reasoning is implemented on top of the TRREE engine. TRREE[17] stands for 'Triple Reasoning and Rule Entailment Engine'. The TRREE performs reasoning based on forward-chaining of entailment rules over RDF triple patterns with variables.
The semantics used is based on R-entailment (ter Horst 2005) with the following differences:

- Free variables in the head of a rule (without a binding in the body) are treated as blank nodes. This feature can be considered '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;
- the [cut] operator can be associated with rule premises, the TRREE compiler interprets it like the ! operator in Prolog;
- Two types of inconsistency checks are supported. Checks without any consequences indicate a consistency violation if the body can be satisfied. Consistency checks with consequences indicate a consistency violation if the inferred statements do not exist in the repository;
- Axioms can be provided as a set of statements, although those are not modelled as rules with empty bodies.

[17] http://www.ontotext.com/trree/index.html

The TRREE can be configured via the rule-sets parameter, that identifies a file containing the entailment rules, consistency checks and axiomatic triples. The implementation of TRREE relies on a compile stage, during which custom rule-sets are compiled into Java code that is further compiled and merged in to the inference engine.
The edition of TRREE used in SwiftOWLIM is referred to as 'SwiftTRREE' and performs reasoning and query evaluation in-memory. The edition of TRREE used in BigOWLIM is referred to as 'BigTRREE' and utilises data structures backed by the file-system. These data structures are organized to allow query optimizations that dramatically improve performance with large datasets, e.g. with one of the standard tests BigOWLIM evaluates queries against 7 million statements three times faster than SwiftOWLIM, although it takes between two and three times more time to initially load the data.

Datatypes

Geo-spatial Extensions

BigOWLIM has special support for 2-dimensional geo-spatial data that uses the WGS84 Geo Positioning RDF vocabulary[18] (World Geodetic System 1984). Special indices can be used for this data that permit the efficient evaluation of special query forms and extension functions that allow:

- locations to be found that are within a certain distance of a point, i.e. within the specified circle on the surface of the sphere (Earth), using the nearby(...) construction;
- locations that are within rectangles and polygons, where the vertices are defined using spherical polar coordinates, using the within(...) construction

RDF Rank

RDF Rank is an algorithm that identifies the more important or more popular entities in the repository by examining their interconnectedness. The popularity of entities can then be used to order query results in a similar way to internet search engines, such as how Google orders search results using PageRank4.

Full text search

Two approaches are implemented in BigOWLIM, a proprietary implementation called 'Node Search', and a Lucene-based implementation called 'RDF Search'. The two approaches are collectively referred to in this guide as 'full-text indexing' and both of them enable OWLIM to perform complex queries against character data, which significantly speeds up the query process. To select one of them, one should consider their functional differences

Basic data types

Currently BigOWLIM doesn't maintain additional datatype-specific indices. Nnumeric datatype indexing is at TODO list for a near future implementation.

Programming with OWLIM

Sesame openRDF framework

OWLIM is built around the RDF data model classes from Sesame and for this reason Sesame is the preferred API to use and the most efficient. OWLIM uses the RDF Data Model classes throughout. The SAIL component (Storage And Inference Layer) contains the classes and interfaces for accessing various storage and inference implementations in a standard way. OWLIM is implemented as a SAIL plug-in to the Sesame framework. At a higher level, the Repository API provides uniform application layer access to Sesame and includes methods for loading/exporting RDF data, preparing and executing queries and so on. The framework includes a console application, a command-line utility for various administration tasks, such as creating/deleting repositories, importing/exporting RDF data, etc.

[18] http://www.w3.org/2003/01/geo/

Jena adapter

The BigOWLIM Jena adapter is essentially an implementation of the Jena DatasetGraph interface that provides access to individual triples managed by a BigOWLIM repository through the Sesame API interfaces. It is not a general purpose Sesame adapter and cannot be used to access any Sesame compatible repository, because it utilises an internal BigOWLIM TRREE API. The adapter comes with its own implementation of a Jena 'assembler' factory to make it easier to instantiate and use with those related parts of the Jena framework. Query evaluation is controlled by the ARQ[19] engine, but specific parts of a Query (mostly batches of statement patterns) are evaluated natively through a modified generator plugged into the Jena runtime framework. There is no Jena support for SwiftOWLIM currently.

ORDI

The Ontology Representation and Data Integration (ORDI[20]) framework is an open-source ontology middleware developed in Java. The main advantage for accessing OWLIM through ORDI is that the triple-set data structures are exposed.

3.4 Distribution and centralization

The centralization/decentralization of storage should be discussed in the context of an overall ebbits systems architecture. Our assumption here is that the ebbits solution should support a range of alternatives considering both centralized and distributed architectures for data management and the control of resources.

We can think of two extremes with respect to the distribution of ebbits and storage facilities and data.

In a fully distributed approach, storage is distributed to the leaves in the network topology. This means that all data and events are kept as close to their originating source as possible.

Pros

- Data is raw (reduced risk of post processing errors)
- Everything is known about a device
- No dependency on central components

Cons

- Devices may be constrained concerning storage capacity
- If need to correlate different devices, reliable time synchronization is needed. This may require an external time server.
- Global data analysis becomes more complex.

In a highly centralized storage alternative, sensor data and events are propagated from leaf nodes (e.g., devices) to one or more central components in the architecture, say ebbits application servers.

Pros

- This will provide for an integrated view of all data and events on a system/global level. This may facilitate performance efficiency in mining and data analysis, e.g., in traceability applications.
- A centralized architecture will facilitate scalability and security.

Cons:

- A centralized solution may impede reliability, although measures can be taken to improve e.g, the reliability and security of storage.

[19] http://jena.sourceforge.net/ARQ/
[20] http://www.ontotext.com/ordi/

- Potential data loss, since data & events may have gone through multiple stages of filtering and/or aggregation.

As in any systems architecture design there are several alternative solution in between these two extremes.

Below we sketch an initial architecture for data fusion and event management in ebbits. This is intended as a context in the further analysis of storage distribution in the system.

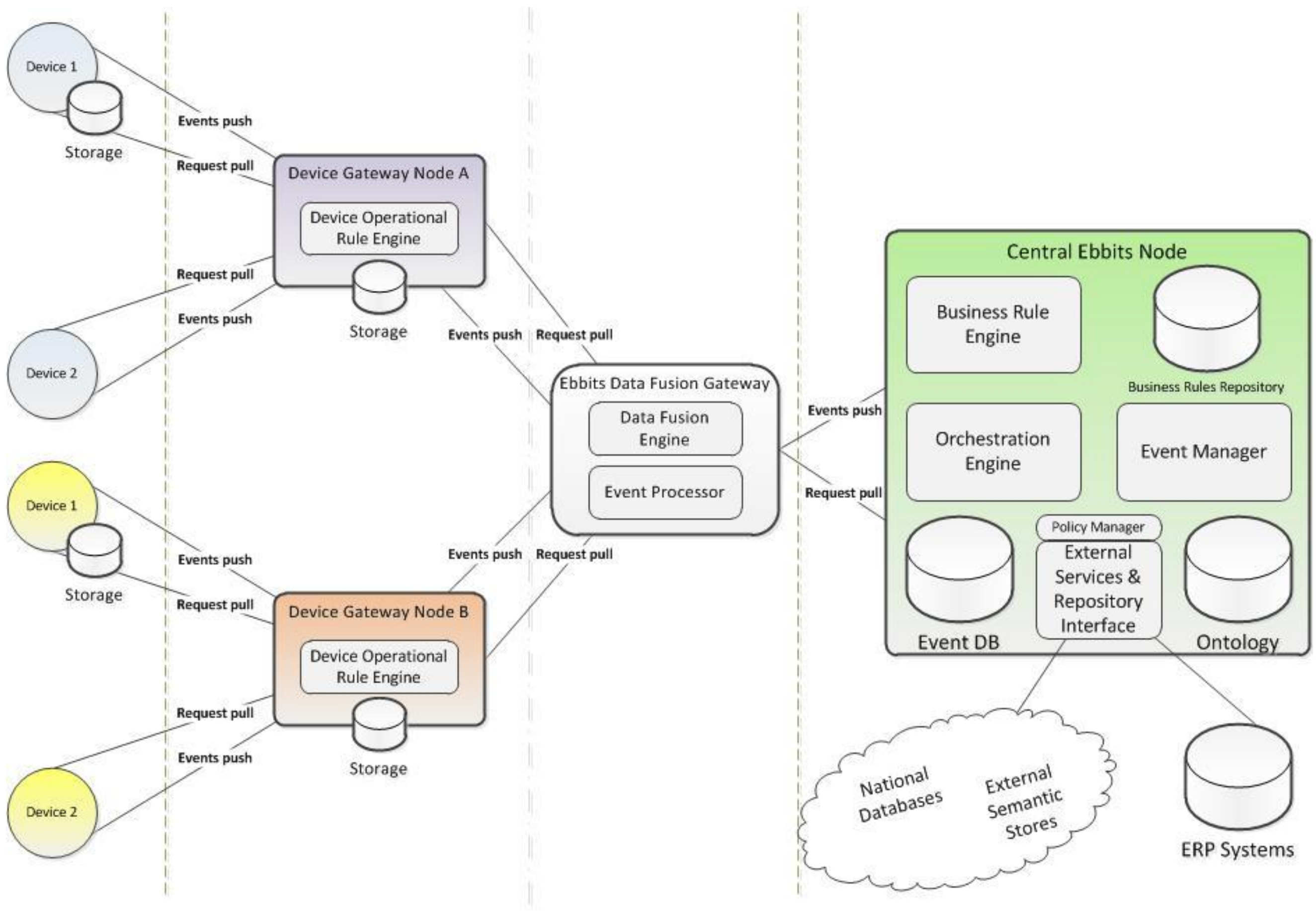

Figure 5: ebbits data fusion architecture

The architecture perspective here is a functional component model, emphasizing the flow of data/events and control (it could also be projected as a layered architecture). Device components (layer) at the left (bottom) and the and Business rules and services components at the far right (top).

Initial component descriptions

Devices 1, 2These devices can be sensors, actuators or even subsystems spread over the ebbits physical environment. For example, it can be a temperature sensor, an RFID reader or some sensor measuring mechanic movement (e.g. pig feeder or roller rotate). Devices generate physical events such as a new temperature value (*stimuli*). Ebbits device storage occurs as close as possible to the device itself. All data kept close to the device, in order to facilitate traceability capability in the ebbits architecture

- **Device Gateway Node A & B**
  The ebbits device gateway node can be based on a PDA or laptop. It enables rule assessment derived from its associated network devices as well as performs rule execution on the gateway node level. Each gateway node can be extended with its own storage. This storage can be used for caching events, etc. in the case of network failures and thereby enabling store and forward. Device gateway nodes combines physical stimuli into device events such as "light is turned on".
  Device Operational Rule Engine

  Each gateway will have a rule engine intended to run a set of device specific rules, execute it locally. For example, not allow the temperature on a device 1 to exceed 5 degrees.

- **Ebbits Data Fusion Gateway**
  The ebbits data fusion gateway has the aim to fuse data that are gathered via a number of ebbits environment's device gateway nodes. Alternatively, the device gateway nodes and the data fusion gateway can be implemented within the same operating platform and thereby enhance the local processing performance. The data fusion gateway combines events from one or several devices and creates application events.

  Data Fusion Engine
  The data fusion engine processes (e.g. aggregates and filters) data from sensors and devices, taking time into consideration. The fused data is further sent to the event processor component.

  Event Processor
  The event processor consumes events and data and creates new application events according to its event management logic. The event processor will dispatch events to the ebbits central node.

- **Central Ebbits Node**
  The central ebbits node is intended to run on high performance machines in the ebbits architecture. By so it will be able to offer the computer power needed to support more intelligent components. This central node can be modelled and positioned as more centrally within an ebbits environment, e.g. at the ebbits service provider. It requires a stable communication with external resources and information providers.

  Business Rule Engine
  This rule engine process a set of business rules defined by the ebbits platform user and describes the intended work flow organisation of the specific domain. Business rules are mapped to services via the orchestration engine. The rule engine uses its own repository. The business rule engine combines one or more application events into a business event which is forwarded to external business system.

  <u>Business Rule Repository</u>
  Here the business rule engine stores and retrieves the set of rules.

  Orchestration Engine

The orchestration engine performs tasks on the request pulls over the ebbits network. This can partly be configured by the business rule engine and partly manually through an interface.

Ontology Database
Here the device ontology is stored for use within the ebbits central node.

Event Manager
The ebbits event manager handles events that are broadcasted throughout the network architecture. It deals with events processed on all levels in the ebbits platform and provides event management to external parties.

Event Database
Provides a chronological and/or source-based log for all events, intended to support data mining and traceability analysis.

[Policy Manager] + External Services & Repository Interface
The policy manager provides access controls by handling communication to external resources. It mediates access to the different services/repositories involved in the ebbits global system.

ERP Systems
For example, SAP, COMAU, etc.
National Databases
For example, regulatory databases for national agriculture control.
External Semantic Stores
For example, a semantic store in a different ebbits node such as an Event database.

3.4.1 Distribution and centralization aspects at multi-sensor data fusion

Some contents from this section have been published in the ebbits Deliverable D5.1.1, Section 5 "Multi-sensor Data Fusion" already.

Sensing

In ebbits project, sensing is a relevant technological approach to drive intelligent service structures. In particular, mobile sensing can be leveraged to monitor many products (perishable goods, auto parts etc) during the end of the life-cycle. Mobile sensing opens new possibilities for data collection through different carrier medium, i.e. human being, robot, vehicle etc wearing or carrying mobile sensors whose data can be sent over a wireless communication channel. In particular, the deployment of smart phones as sensor nodes facilitates applications that allow for observations of phenomena or events, which previously were hard to perceive or even impossible. This is known as crowd sourcing. In the traceability scenario the subject to observe are pigs, in fact 'from farm to fork'. Due to transports, climatic changes, or improper nutrition etc animals are susceptive to momentous diseases. Hence, to avoid the widespread of disease pigs wear mobile sensor nodes pigs, so that at an early stage a (contagious) disease can be identified. Brownfield development in manufacturing site, i.e. plant is already running and comprises several machines, it is more challenging (Hopkins and Jenkins 2008), as you will need to deploy new sensors and actuators in the immediate presence of existing (legacy) systems. In this sense, it is desirable to deploy mobile sensor nodes as an overlay system to the existing infrastructure so that the risk of introducing technical problem to the running production system can be minimized.
Dealing with massive amount of sensor data, requires an intelligent data management that include aggregating, filtering, and joining data into useful information. This approach is known as Multi-sensor data fusion. Multi-sensor data fusion is a technology that has been formalized since 70s specifically in military applications. Many architecture, model, and algorithms have been developed addressing various data fusion applications. JDL model is the most used and well known model (Liggins, Hall et al. 2009). It provides a partition of sensor fusion's

functions. However, it does not describe any process model and architectural design. Thus, in ebbits, it is necessary to complete this model from different architectural views to facilitate a common understanding of sensor fusion processes for ebbits domains.
The most mature area of the JDL model is level 0-1 processes such as target tracking, position, velocity determination, and object classification (friend or foe). All of these applications tried to estimate the certainty of information obtained from sensor data. Nonetheless there is still no general solution that is able to overcome challenges such as object densities, rapid movement, and signal propagation. Many algorithms in this level are required in ebbits scenarios for instance, object tracking is useful for tracking animals in farms and goods in factories, increasing the certainty of wireless transmission and sensors readings in places with harsh conditions. Algorithms have been maturely used in this level includes: filtering algorithms, aggregation and compression. Level 2 and 3 fusions are dominated by knowledge based such as rule based, fuzzy logic, intelligent agents, and Bayesian beliefs. Although these areas are quite promising to provide an intelligent information, unfortunately these techniques are still immature and do not provide any stable operational systems (Liggins, Hall et al. 2009). The main challenges in this are establishing a common and reliable knowledge base and representing it uniformly, many works on fuzzy logic have shown promising results, though. Another challenge that ebbits will contribute to this area is that it aims at massive scale of distributed and heterogeneous systems that have diverse knowledge representations.
Ebbits requires defining a new framework that reflects process model as well as different architectural views of information fusion. Furthermore, the ebbits must also define a concept to handle diverse knowledge representations distributed in the internet of things. An example where level 2 and 3 processing will be needed in ebbits is e.g.: for inferring intelligent context for energy savings purposes.

Control Management

Many multi-sensor fusion models have included control theory that discusses relationship of sensing and control as well as sensing-control loop for performance assessment of the system. Up to now, the main challenges in this area are to model task objectives, manage resources based on the objectives, and provide information that satisfies the decision makers' needs. ebbits also aims at a self regulating system, which need an automatic control and real time system assessments. On the other hand, if the control and assessment is done manually through human intervention, the human computer interaction concepts must also be defined.

Dorf and Bishop defines *control system as an interconnection of components forming a system configuration that will provide a desired system response* (Dorf and Bishop 2008). Industrial control system covers several standard solutions that have been used in automatic manufacturing system such as Supervisory Control and Data Acquisition System (SCADA), Distributed Control System (DCS), Advance Process Control, smaller controller units such as Programmable Controller Logic (PLC), and *Integrated Control and Safety System* (ICSS). SCADA usually supervises and coordinates an entire site or even multiple distributed sites. However it does not control in real time (it does not utilize any real time operating system). DCSs are dedicated for controlling automated processes of batch productions. DCSs are intended to distribute intelligence in production plants by coordinating PLCs that control equipments independently. PLC is a digital computer used for automating electromechanical processes that evolves from automotive industry in the 70s(Zhang 2008). PLC is now widely used in almost any production automation areas including slaughterhouses and feeding systems in farms. Safety controls in manufacturing equipments are regulated by *Integrated Control and Safety System* (ICSS). The ICSS usually includes three types of safety system such as Process Shutdown System, Emergency Shutdown/Depressurization, and Fire-Gas Safety System.

Wireless sensor and actuator network (WSAN) propose a promising technology to replace and complete the existing technology in the manufacturing plans as well as food production because wireless offers a better flexibility in deployment than wired solution. However, wireless communication introduces higher complexity in communication means as it is more vulnerable against interferences. There have been many theoretical research and work being done to increase the wireless network resilience. However, there has only been a limited amount of practical work in the field. Thus, further studies applying wireless networks in the manufacturing plants and food production in context of ebbits project should be conducted. The existing control theories always assume perfect network reliability which is not the case in wireless communication. Therefore, there is in ebbits a need to introduce communication parameters within the existing control theories. Energy supply in WSAN is an important factor. In this section, many works have been explored to conserve energy supply of WSAN by controlling the nodes and network behaviors. In ebbits we need to study the communication patterns in order to conserve the energy supply effectively.

Deriving Knowledge from sensor data

There exist many approaches for deriving knowledge from sensor data. One of the techniques is by tagging sensor data with metadata. The tagging process is defined in sensor language such as SensorML[21] and EEML[22]. Metadata can be the processed to derive the meaning of the data. Processing the metadata can be done by probabilistic inference algorithms such as Bayesian and Dempster-Schafer, fuzzy logic, or as simple as using rule engines. The main drawback for rule engines is that the rules are not flexible meanwhile in the real world there exist exceptions and exceptions of an exception which make rules invalid. Thus, in ebbits we would like to investigate the business rule engines, that are commonly used in industry and hybrid solution between rule engine and probabilistic inference approaches. WS-BPEL[23] provides a standard language that is used by rule engines in SOA.
Mining data and information from distributed sources can be done by using different approaches e.g.: approaches evolving from database domain, web, and heterogeneous sources. New approach in database domain is called *Dataspace* (Franklin et al., 2005) that increases semantic cohesion over time by assuming that different parties provide mappings of how different knowledge representations can be linked. Thus providing an unified architecture for reference reconciliation, schema matching and mapping, data lineage, data quality and information extraction. In web domain semantic web allows software agents / crawler to query information from distributed sources examples of this approach are SemaPlorer (Schenk et al. 2008) and SearchWebDB (Tran et al. 2008). Coming from web 2.0, many mesh up platforms such as yahoo pipe are used to aggregate information and services. Aletheia[24], a German national project for harmonizing product information uses semantic abstraction and human interference for resolving conflict. OKKAM[25], an FP7 EU ongoing project, tries to connect the corporate structured enterprise data with unstructured data such as product documentations that are written from humans.

Data populated from distributed sources, documents and any other sources might present erroneous knowledge. There are 2 ways to deals with this kind of data, first to purge erroneous state and secondly by finding ways to work with it by modifying the reasoning approach (Huang et al 2005). Erroneous might be caused by inconsistency, incompleteness, and redundancy. Another drawback of ontology is the inability to represent and reason upon uncertainty. There exist few works trying to present uncertainty e.g.: OWL-DL and PR-OWL (Probabilistic information in OWL).

Context aware computing is a branch of computer science that uses sensor data to change the application behavior. Modeling context can be done by Key Value Models, Markup Scheme Models, Graphical Models Object-Oriented Models, Logic-Based Models, and Ontology Based

[21] http://www.opengeospatial.org/standards/sensorml
[22] http://www.eeml.org/
[23] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[24] http://www.aletheia-projekt.de/
[25] http://www.okkam.org

Models (Strang and Linnhoff-Popien, 2004). Hydra follows a hybrid approach that models low level data through key-value and high level context through ontology model. Hydra introduced three types of context: data, semantic, and application.

4. Ebbits use cases analysis

This section provides an analysis of the ebbits use cases. The use cases imply requirements on all of the aspects of semantic stores.

4.1 Scalability and queries

From the ebbits requirements in the Deliverables D2.1 and D2.4 it follows that an RDF store used in ebbits has to provide query and reasoning functionality about large knowledge bases. For this reason a system providing distributed hierarchical knowledge bases, with in-memory processing of the required subset seems reasonable for a high performance. In the following we describe technologies that tackle especially those challenges and could be candidates for prototypes within the ebbits project.

Sesame is an API for storing and querying RDF data. Figure 6 shows the class diagram of the RDF data model.

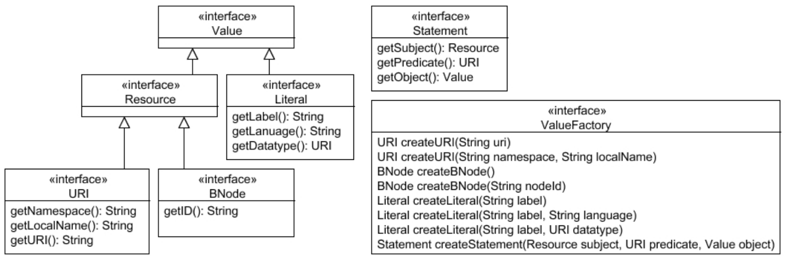

Figure 6: Class diagram of the RDF data model

Sesame has two main communication interfaces: the Sail API and the Repository API. The Storage and Inference Layer (Sail) API is a low level system API for RDF stores and inference engines. Its purpose is to abstract from the storage details, allowing various types of storage and inference to be used (Aduna 2010a).

The Repository API is a higher level API and is meant to be the main API that people can program against. It offers various methods for uploading data files, querying, and extracting and manipulating data. It comes in two flavors: local and remote.

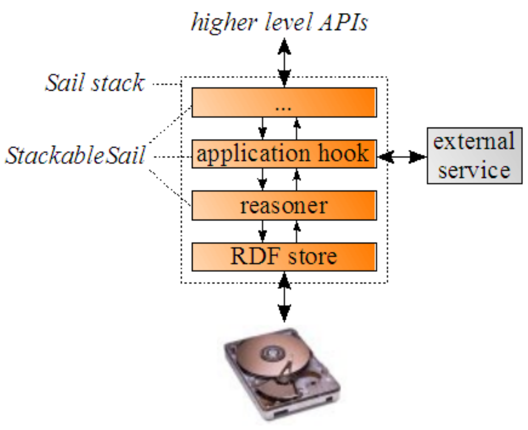

Figure 7: Example of a Sail stack

Sails can be stacked on top of other Sails. By stacking a Sail on top of another, all calls for the bottom Sail will pass through the Sails that are on top of it (see Figure 7). This architecture is used for a whole range of applications: access control, pluggable inference engines, hooks to external services, etc. A concrete example is a mixed forward-/backward-chaining inference engine, which will wrap transactions to do its forward-chaining work upon commit and extend queries to do its backward-chaining work.

For scalability reasons, all data that is extracted from a Sail object is returned in the form of (forward-only) iterators. This allows one to fetch the entire set of stored statements, even if the set is too large to be stored in main memory.

In the Aletheia research project a Sail implementation used the Lucene API as an external service for indexing literals in the RDF triples stored in a repository. This allows to combine a full text search over RDF triples and any other graph query in SPARQL in a similar way as described in (Minack et al. 2008).

Through the Sail mechanism, Sesame claims to support federated repositories in a distributed environment. The federation "distributes sections of the query to different members based on the data contained in each of the members. These results are then joined together within the federation to provide the same result as if all the data was co-located within a single store" (Aduna 2010b).

Distribution imposes not only new challenges to storage and querying but also to reasoning. The recent publication (Urbani 2010) addresses this issue with a MapReduce frame-work for distributed computation of the closure of an RDF graph under the OWL Horst semantics. The WebPIE inference engine is built on top of the Hadoop platform, deployed on a cluster of 64 machines and evaluated with datasets containing up to 100 billion triples. The results have shown that the system is scalable and vastly outperforms competing systems when comparing supported language expressivity, maximum data size and inference speed (Urbani 2010).

4.2 Reasoning

User requirements from D2.4 report were analysed to identify these, which can have some relation to a triple store capabilities. In the table below, these requirements are presented, together with relevance to some of analysed properties of triple stores analysed in previous chapters.

Table 8: Possible Triple Store utilisation in relevance to the User Requirements

Req. No.	Req. Description	Relevance to triple stores. Possible utilisation of a selected triple store to fulfil the requirement.
functional requirements		
17	Semantic relationships between data. Currently, any data is stored in a simple database. Hence, data is available, but cannot be interrelated intelligently.	Data stored in a triple store can be used for reasoning and querying. The cost is a need for semantic description of existing data. Data already stored in databases can be usually described automatically or semi automatically
18	Aggregating collected sensor data at a central point. The aggregation of collected data is important for analysing the data.	Federation support in triple stores enables storing data on several places while still analysing it as one
19	Farmers are able to retrieve optimized models from research. Farmers are willing to share data if they could get something in return such as models to optimize feeding process.	Named graph support in triple stores enables using several research models together with user data without mixing them
20	System can feed the farms data to	Export and import functionality of triple

	research. Most of the farming models are developed by research organizations, universities etc.	stores, together with named graph support enables combining several independent models together.
70	Support system for comparing different energy consumption among plants and corresponding processes. Management would like to learn from other plants if they use energy more efficiently.	Federation support in triple stores enables remote usage of models. Named graph support together with import enables reasoning and querying above combined resources
71	Summary of energy related information at operational level for supporting management level optimizing energy use. Operational management needs a summary of energy related information that help them making decision to optimize the energy usage.	Querying of triple store is a possible solution
72	Recognition of energy wasting behaviours. Help decision makers to optimize energy usage.	Reasoning or querying in ontologies can help to identify these behaviours
73	Items need to be traced within an enterprise. Goods and items need to be traced within one farm or enterprise.	Support for location data is needed in triple store to enable location aware querying
75	Information needs to be described in a standardised way. Enterprises working in the same sector adapt different ways to describe the input, the production processes, and the output; thus it will not be possible to communicate information either to providers or to consumers.	Triple stores can store heterogeneous data interoperable via semantic relations between them
77	Associate meta-information to items. In parallel to the actual lifecycle (grow up of the animal, feeding, butchering, transportation, selling, consuming) there exists additional information such as the amount of food, medication an animal has had, the energy for the production and transportation, that needs to be acquired and associated with the (bits and pieces of) animal.	Several ontologies can be used at once in a triple store; reasoning can help to overcome possible heterogeneities between multiple semantic data sources.
82	Support fuzzy or probability concepts for reasoning. There is no reasoning algorithm that is able to solve any kind of cases.	More investigation of reasoning capabilities is needed for that matter. However, some triple stores are modular enough to use any external reasoner. Probabilistic reasoners (e.g., Pronto) can be evaluated for performance and included into the solution.
84	Different views on the device ontology. It should be possible to present a developer user with different perspectives on the device ontology, depending on that users functional needs (e.g., a services perspective, device category perspective. etc.).	Triple stores use recommended W3C standards to enable use of external tools for visualisation of ontologies

86	Protection of System Integrity. In order to prevent an inexperienced user to cause malfunctions by changing system configurations, the middleware should monitor, analyse and, if necessary, prevent or give notifications about faulty changes.	Transactions style changes has to be supported in the triple store
87	Handling of different device versions in device ontology. The device ontology should be able to handle different versions of a device.	Time snapshots of ontologies and selective work with subset of these has to be supported in the triple store
93	Download and harmonisation of third party device ontologies. Device ontological models describing devices, which will be provided by manufacturers or third parties, should be automatically downloaded (updated) and harmonised to ensure the same ontological view. Formal definition of ontologies should be realised using the world wide accepted formats, recommended by W3C, such as RDF, OWL, OWL-S. Hydra open requirements	W3C standards and recommendations has to be used in triple stores
103	The system should allow the correlation of information emerging from several sources. In order to easily analyse information, the system should allow for the correlation of information from different sources on a farm or enterprise.	Reasoning support should enable a correlation of concepts in different ontologies
105	Aggregate data from various data bases and sources. Information will be stored in several places, but needs to be combined in some place and assigned to the actual product or entity.	Data should be imported into a triple store using recommended formats and formalisms
108	Different Views on the Data is necessary. We need services that provide different views on the data cloud by combining data from different sources.	Querying of ontology and ontology visualisation can support these
non functional requirements		
2	The ebbits should be able to handle massive number of devices. The future use cases of eBBits need to handle massive number of devices and applications within and cross enterprises, i.e. ci. 300-1000 in a manufacturing plant and 500 in a farm.	Scalability has to be an important measure when selecting a triple store to be used
22	Resilience and adaptable to environment condition changes. Environmental changes such as lighting, temperature affect the results of manufacturing process. So far machines are tuned manually by technicians. Adapting to environmental condition can lead to reducing energy consumption	Time dependent values has to be supported in selected triple stores

	e.g.:reduce heater temperature when it's warm outside.	
24	Filtering to Obtain relevant Information. Too much information overwhelm farmers while making decisions.	Efficient filtering using queries is supported in triple stores. Duplicate information filtering can also help
61	Scalable solution (scale up and scale down). Adjustment to desired number of production, require to add or reduce machines.	Scalability enables triple stores to work with different amounts of data. From hundreds of triples on a HW resource-weak systems up to billions of triples on distributed multi-core high on RAM systems
110	End-users need to be able to managment their distributed data. Farmers want to manage their distributed data, because today they have no full control of data.	Tools for managing triple store should be available

4.3 Distribution and centralization

In a massive distributed environment as foreseen in ebbits, access to resources such as services and device features may be radically different, ranging from human-only access (e.g. screens and buttons), to exclusive M2M (machine-to-machine) communication over standard protocols and wireless transports. This wide array of access mechanisms might be acceptable for the individual device manufacturer, but not for the developer who wants to build solutions based on a vast number of devices from different manufacturers with obscure access protocols, that for the most remain proprietary or unknown. This makes it almost impossible for existing devices to communicate and to interchange useful information and/or commands structures, without the direct involvement of the manufacturers of the devices in one way or the other. The way to access intelligent services across a distributed network is for the ebbits platform (see Figure 8) to create a ubiquitous communication infrastructure that automatically and dynamically connects to sensors and devices in the physical world in e.g. manufacturing facilities or in private smart homes. It further connects to mainstream backend information systems, public authentication systems and regulatory information sources using semantic web services and finally connects to human users in dispersed geographical locations (e.g. professional users in technical support, field service and other business environments as well as ordinary consumers in shops or at home).

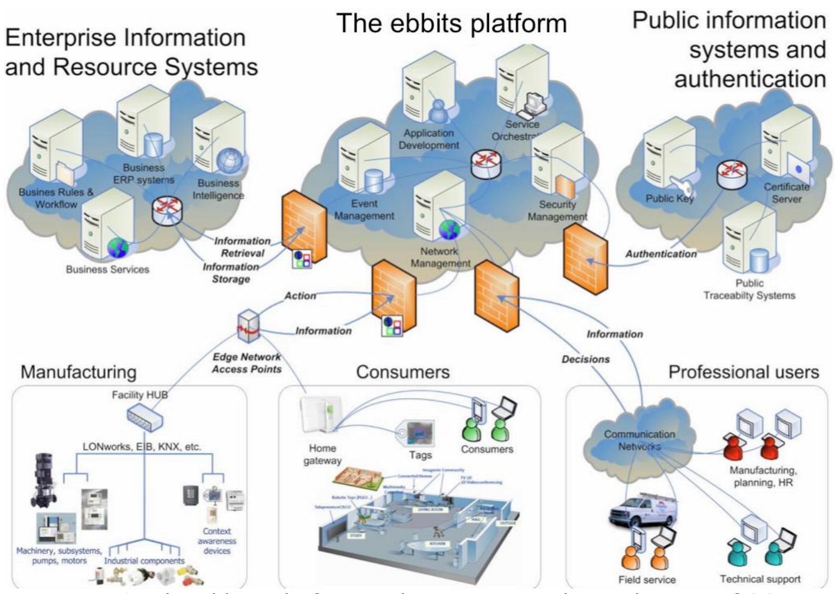

Figure 8: The ebbits platform with a prototypical actualization of SOA

The ebbits platform consists of subsets of production servers for data management, event management, security, application execution and communication where all servers are to interoperate in an open architecture on the basis of web services. The ebbits SOA is completely platform agnostic and scalable. The implementation of SOA for product lifecycle management and manufacturing should follow ISO 10303 specifically the following application protocols:

Manufacturing APs:

- AP 219, Dimensional inspection information exchange
- AP 223, Exchange of design and manufacturing product information for cast parts
- AP 224, Mechanical product definition for process plans using machining features
- AP 238, Application interpreted model for computer numeric controllers
- AP 240, Process plans for machined products

Life cycle support APs:

- AP 239, Product life cycle support
- AP 221, Functional data and schematic representation of process plants

5. Conclusions

This Deliverable forms the starting point for Task 4.1 of the ebbits project. It presents the state of the art of semantic stores and brings it into relation to the ebbits use cases in order to identify the gap to be filled during the ebbits project in Work Package 4 "Semantic Knowledge Infrastructure".

Section 3 presents the state of the art of semantic stores with the aspects (1) scalability of RDF stores and query performance, (2) reasoning, and (3) distribution and centralization strategies. All three aspects are picked up in Section 4 again when they are brought into relation to the ebbits use cases and requirements.

Scalability of RDF stores and query performance is a subject which has been investigated in the research community in studies to compare the performance of practical implementations with given datasets. The W3C maintains a collection of RDF store benchmarks and a subset that is relevant for the ebbits project, has been presented. In several test settings, Virtuoso performed best, followed by Sesame. Since OWL reasoners can be connected to state of the art RDF stores, an approach for OWL reasoner benchmarking has been presented. As it has been expected, current implementations offer a good starting point but contain bugs in the implementation as well as unsolved conceptual problems. Considering ebbits use cases and requirements, an RDF store used in ebbits has to provide query and reasoning functionality about large knowledge bases at high performance. Technical candidates to meet those requirements seem to be systems that provide (1) distributed hierarchical knowledge bases with query distribution and result fusion, with (2) in-memory processing of the required subset, and with (3) a modular architecture that allows for the integration of existing components, e.g. high performance full text search engines.

The overview on the state of the art of reasoning in RDF stores discusses two selected RDF stores in detail. They have been selected from a set of the most known triple stores containing BigOWLIM, SwitftOWLIM, Bigdata, AllegroGraph, OntoBroker, Sesame, and Jena. The selected triple stores are BigOWLIM and AllegroGraph. BigOWLIM is capable of efficiently working with up to 20B triples and AllegroGraph even with more than 20B triples. Both appear suitable in the context of the ebbits requirements and use cases. Ebbits semantic subsystems will be developed in a modular way, so that usage of different triple stores will be possible without too much effort needed for configuration or reimplementation.

Distribution and centralization aspects are an important part of the considerations in the ebbits architecture. Those decisions also have effects on the nature of the used RDF stores. A section about those aspects starts with a general discussion of distributed vs. centralized systems, bringing the ebbits data fusion architecture into context as a system between those two extremes. References to other ebbits Deliverables bring the current Deliverable into context and provide an overview, while avoiding the pure repetition of concepts already explained in those Deliverables. The aspects discussed cover sensing, control management, and the derivation of knowledge from sensor data that is stored in an ebbits RDF store.

6. References

(Aduna 2010a) Aduna. User Guide for Sesame 2.3. Available online at http://www.openrdf.org/doc/sesame2/2.3.2/users/ch03.html. 2010.

(Aduna 2010b) Aduna. Federation Sail. Available online at http://www.openrdf.org/doc/alibaba/2.0-beta4/alibaba-sail-federation/index.html. Last Published 2010-12-23.

(Baader 2009) Franz Baader: Description Logics. In: Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009, LNCS 5689, Springer–Verlag, 2009, pp. 1–39.

(Baader and Nutt 2002) Baader, F., Nutt, W.: Basic Description Logics. In: The Description Logic Handbook, Cambridge University Press, 2002, pp. 47-100.

(Bail, Parsia and Sattler 2010) Samantha Bail, Bijan Parsia and Ulrike Sattler. JustBench: A Framework for OWL Benchmarking. In Proceedings of the 9th International Semantic Web Conference (ISWC2010). 2010.

(Bizer and Schultz 2009) Chris Bizer and Andreas Schultz. Berlin SPARQL Benchmark Results. Available online at http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html. 2009.

(Bock et al. 2008) Jurgen Bock, Peter Haase, Qiu Ji, Raphael Volz: Benchmarking OWL Reasoners. In: Proceedings of the ARea2008 Workshop. CEUR Workshop Proceedings, vol. 350., 2008.

(Davies et al. 2006) John Davies, Rudi Studer, Paul Warren: Semantic Web Technologies. Trends and Research in Ontology-based Systems. John Wiley & Sons, Ltd, Chichester, England, 2006.

(Dorf, R. and R. Bishop 2008) Modern control systems, Pearson Prentice Hall.

(Fielding, R. 2000) Representational state transfer (REST). Chapter 5 in Architectural Styles and the Design of Networkbased Software Architectures, Ph. D. Thesis, University of California, Irvine, CA, 2000.

(Franklin et al. 2005) Franklin, M., Halevy, A., Maier, D.: From Databases to Dataspaces: A new Abstraction for Information Management, Sigmod Record, ACM, 34(4):27, 2005, p. 33

(Guo et al. 2005) Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 2005, pp158-182.

(Hopkins, R. and K. Jenkins 2008) Eating the IT Elephant: Moving from Greenfield Development to Brownfield, IBM Press.

(Huang et al. 2005) Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'05, 2005.

(Lee et al. 2008) Chulk Lee, Sungchan Park, Dongjoo Lee, Jae-won Lee, Ok-Ran Jeong, Sang-goo Lee: A Comparison of Ontology Reasoning Systems Using Query Sequences. In: ICUIMC '08, Proceedings of the 2nd international conference on Ubiquitous information management and communication, ACM New York, 2008, pp. 543-546.

(Liggins, Hall et al. 2009) Liggins, M. E., Hall, D. L., Llinas, J.: Handbook of Multisensor Data fusion, Theory and Practice. Boca Raton, FL, CRC Press, 2009.

(Ma et al. 2006) Ma, Li and Yang, Yang and Qiu, Zhaoming and Xie, Guotong and Pan, Yue and Liu, Shengping. Towards a Complete OWL Ontology Benchmark. In:

The Semantic Web: Research and Applications. LNCS 4011. Pages 125-139. 2006

(Minack et al. 2008) Enrico Minack, Leo Sauermann, Gunnar Grimnes, Christiaan Fluit and Jeen Broekstra. The Sesame Lucene Sail: RDF Queries with Full-text Search. Technical Report. Available online at http://nepomuk.semanticdesktop.org/xwiki/bin/download/Main1/Publications/Minack%202008.pdf. 2008.

(Pearl 1997) Judea Pearl: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., 1997.

(Schenk et al. 2008) Schenk, S., Saathoff, C., Baumesberger, A., Jochum, F., Kleinen, A., Staab, S., Scherp, A.: SemaPlorer - Interactive Semantic Exploration of Data and Media based on a Federated, 2008.

(Schmidt 2009) Michael Schmidt, Thomas Hornung, Georg Lausen, Christoph Pinkel. SP2Bench: A SPARQL Performance Benchmark. In Proc. ICDE 2009, Shanghai (China).

(Schmidt 2010) Michael Schmidt. Foundations of SPARQL query optimization. PhD thesis. Universität Freiburg. 2010.

(Serrano et al. 2007) Serrano, J. M., Serrat, J., Strassner, J.: Ontology-Based Reasoning for Supporting Context-Aware Services on Autonomic Networks. In: Proc. of ICC '07, IEEE International Conference on Communications, 2007, pp. 2097-2102.

(Strang and Linnhoff-Popien, 2004) Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey, In: Proc. of Workshop on Advanced Context Modelling, Reasoning and Management, 2004.

(ter Horst 2005) 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, 2005, pp. 668-684.

(Tran et al. 2008) Tran, T., Wang, H., Haase, P.: SearchWebDB: Data Web Search on a PayAsYouGo Integration Infrastructure, Technical Report, Universität Karlsruhe (TH), 2008.

(Urbani 2010) Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. Proceedings of the Seventh European Semantic Web Conference. LNCS. 2010.

(W3C 2010) W3C. RDF Store Benchmarking. Available online at http://esw.w3.org/RdfStoreBenchmarking. 2010.

(W3C Implementations 2010) W3C - OWL Working Group, Implementations. Available online at http://www.w3.org/2007/OWL/wiki/Implementations, 2010.

(Wikipedia) SPARQL at http://en.wikipedia.org/wiki/SPARQL

Semantic data model at http://en.wikipedia.org/wiki/Semantic_data_model

(Zhang, P. 2008). Digital Controller for Industrial Control. Industrial Control Technology, William andrew Inc.**:** 429.