Real-World Reasoning with OWL

Timo Weithöner[1], Thorsten Liebig[1], Marko Luther[2], Sebastian Böhm[2], Friedrich von Henke[1], and Olaf Noppens[1]

[1] Inst. of AI, Ulm University, Ulm, Germany
firstname.lastname@uni-ulm.de
[2] DoCoMo Communications Laboratory Europe GmbH, Munich, Germany
lastname@docomolab-euro.com

Abstract. This work is motivated by experiences in the course of developing an ontology-based application within a real-world setting. We found out that current benchmarks are not well suited to provide helpful hints for users who seek for an appropriate reasoning system able to deal with expressive terminological descriptions, large volumes of assertional data, and frequent updates in a sound and complete way. This paper tries to provide some insights into currently available reasoning approaches and aims at identifying requirements to make future benchmarks more useful for application developers.

1 On Benchmarking OWL Reasoners

Having sufficiently exhaustive knowledge about the influence of the underlying reasoning approach on the practical tractability of a particular ontology is of fundamental importance when selecting an inference engine for a real-world application. By real-world we mean an ontology-based application with an expressivity at least beyond $\mathcal{ALC}$, containing more than thousands of individuals, and an inference response time of less than a second, even in a dynamical setting of frequent ontology updates. For instance, context-aware applications want to offer services to users based on their actual situation. Experiences in the course of operating a context-aware application for mobile users [1] clearly have shown that the quality of such an application hosted on a server significantly depends on the availability of reliable and scalable reasoning systems able to deal with constantly changing data. In order to meet real-world needs a reasoning system also has to offer a sufficiently expressive query language as well as a flexible and efficient communication interface.

Unfortunately, current benchmarks or system comparisons neither draw a clear picture of the landscape of practically tractable language fragments with respect to large amounts of instance data, give valuable insights into pros and cons of different reasoning approaches, identify performance penalties caused by certain language features, nor consider issues such as updates, incremental query answering, or interfaces.

For instance, many benchmarks consist of synthetical generated and sparsely interrelated data using inexpressive ontology languages such as the widely used

E. Franconi, M. Kifer, and W. May (Eds.): ESWC 2007, LNCS 4519, pp. 296–310, 2007.

Lehigh University Benchmark (LUBM) [2]. In case of the LUBM an incomplete query answering procedure exploiting told information about the individuals from a classified TBox is sufficient to answer the given queries correctly. The published RacerPro results [3] heavily rely on this property of the LUBM. The bottom line is that RacerPro shows an impressive performance for solving this ABox benchmark by switching off ABox reasoning. Obviously, this cannot be considered as a meaningful benchmark and it is not surprising that this test suite has let to exceptional performance for almost all inherently incomplete reasoning systems. On the other hand, the University Ontology Benchmark (UOBM) [4], a direct extension of the LUBM in terms of expressiveness, turned out to be much too difficult for most systems to answer correctly within reasonable time. This well known trade-off between tractable and effectively un-tractable ontologies, the so called *computational cliff*, is caused by an increase in language expressivity [5]. A more fine-grained map of the border of effectively tractable ontologies still needs to be practically explored in order to be helpful for developers.

The discussion of inherent drawbacks and advantages of different approaches with respect to diverse application tasks has been largely neglected in recent benchmarks or system comparisons. However, application developers need to be aware of potential trade-offs and a serious benchmark should discuss its results with respect to alternative reasoning approaches.

Another performance related issue deals with the way of feeding the systems with large amounts of data. Our selective tests have shown that for some systems not only the transmission format (RDF/XML or DIG [6]) is of importance, but also the way data is encoded (e. g. deep vs. flat serialization).

A real-world requirement which has not been taken into account in any benchmark so far is concerned with dynamic data. The ABox is not only expected to be the largest part of an ontology but is also subject to frequent changes. In order to serve as an efficient reasoning component within a realistic setting it is necessary to perform well under small ABox updates. First results in this research direction, e. g. [7], need to be evaluated by appropriate benchmarks.

Finally, all benchmark results need to be weighted with respect to soundness and completeness of the underlying inference procedure. Assuming that soundness and completeness is an indispensable requirement for knowledge-based applications — of which we think it is — many of the existing benchmark results are not helpful at all. Some of our randomly selected tests showed that even systems assumed to implement a sound and complete calculus fail on a number of OWL Lite test cases.

Our overall goal is to qualitatively analyze various benchmark suites and results in order to identify requirements for a comprehensive benchmark suite suitable to allow ontology-based application developers to pick the right system for their individual task. In the following section, we compare alternative reasoning approaches. We then (Section 3) analyze existing benchmark suites, discuss corresponding results and compare them with our own tests. As a result we compiled a collection of requirements (Section 4) to make future benchmarks more useful for application developers. Section 5 summarizes our experiences and suggestions.

2 System Analysis

Understanding and interpreting benchmarking results correctly requires to have some insights into alternative processing methods of different system implementations. In the context of reasoning with OWL, or fractions thereof, one can roughly distinguish between four different approaches.

Due to its historical origins the inference calculus implemented in **tableaux-based provers** for DLs is an obvious choice and available via systems like Pellet [8], RacerPro [9], or FaCT++ [10]. They implement a conceptually sound as well as complete approach for which many optimizations are known so far. Unfortunately, complete instance reasoning still requires expensive computations but recent research on elaborated reduction methods [11] show enormous optimization possibilities in this respect.

An alternative, equally sound and complete, approach is to transform an OWL ontology into a disjunctive datalog program and to utilize a **disjunctive datalog engine** for reasoning as implemented in KAON2 [12]. This allows for fast query answering due to well-known optimization techniques from deductive databases such as magic set transformation. A drawback is that this approach does not support nominals and has some performance problems with cardinality restrictions in presence of certain other axioms.

Other systems like OWLIM [13] or OWLJessKB use a standard **rule engine** to reason with OWL. This is fast and easily tunable to different language fragments just by manipulation the rule set. However, this procedure is known to be incomplete and resource consumptive when filled with large amounts of implicit knowledge because of their materialization strategy.

A couple of more or less **hybrid approaches** such as QuOnto [14], Minerva [15], Instance Store [16], or LAS [17] combine an external reasoner (often a tableaux-based system) with a Database system. This enables to process large data volumes due to secondary storage mechanisms. On the other hand, this combination only allows for a very limited language expressivity.

3 Benchmarking Experiences

This section tries to roughly draw a picture of practically tractable OWL repositories with current reasoning systems. This is done by gathering data from different existing as well as own benchmarks. The collected results are reviewed with respect to the system, i. e. the underlying approach, as well as the kind of test ontologies.

A common benchmark for today's DL reasoners is to measure the performance in processing huge ontologies. Ontologies with relatively small and inexpressive TBoxes and ABoxes containing hundreds of thousands or even millions of individuals and relations are predominantly used in benchmarking tests. It is assumed that real world applications will also exhibit the described characteristics.

A selection of such "real world ontologies" (e.g. Gene Ontology[1] or Airport Codes Ontology[2]) which are used for benchmarking can be found in [18].

Nowadays, the Lehigh University Benchmark (LUBM) is the de facto standard when it comes to reasoning with large ontologies [3,19,8,20,21]. But as mentioned before, many reasoners that achieved excellent results when benchmarked with LUBM, failed to reason about other ABox or TBox tests (cf. results for OWLIM and KAON2 from Sections 3.1 and 3.2).

The University Ontology Benchmark (UOBM) [4], extends the LUBM by adding extra TBox axioms making use of all of OWL Lite (UOBM Lite) and OWL DL (UOBM DL). In addition, the ABox is enriched by interrelations between individuals of formerly separated units, which then requires ABox reasoning to answer the given UOBM queries. Not surprisingly, it turned out that incomplete systems now can only answer a fraction even of the OWL Lite queries completely. Only one theoretically sound and complete approach, namely Pellet [8], was able to handle about a tenth of the number of individuals compared to the LUBM. The others failed either due to a timeout or the lack of memory.

These shortcomings motivated us to experiment with a set of tests of a different kind using both existing and newly created benchmarks. In the following, we will present some of these measurements, whereas the aim of these tests was not to simply nominate the fastest or most reliable reasoner. Also, instead of overloading this report with a complete set of all of our measurements we will highlight some results that demonstrate the necessity and requirements for a comprehensive benchmark that goes beyond the LUBM.

In the following, we will present selected results for KAON2 (built 05-12-2005), Pellet 1.3, OWLIM 2.8.3, and RacerPro 1.9.0 which were selected from three out of four different reasoning approaches mentioned in Section 2. FaCT++ was dropped due to a missing query language and all hybrid systems were not appropriate because of their limited language expressivity. We divided the whole process of loading an ontology, any preprocessing as applicable and processing of a query into two separate measurement stages:

Loading and preprocessing the ontology. This stage summarizes the measurements for the process of feeding the ontology into the reasoner and any parsing as required by the interface. Also any preprocessing that is either done automatically or can be started manually is included into this measure. For most of the benchmarks presented in this report this measurement is dominated by the time needed to load the ABox as TBoxes tend to be very small and incomplex.

Query processing. This stage measures the time and resources needed to process a given query and for some systems might also include preprocessing efforts.

Loading of ontologies was repeated three times (discarding them after the first two passes, keeping them after the third). Then the respective queries were repeated

[1] `http://archive.godatabase.org/`

[2] `http://www.daml.ri.cmu.edu/ont/AirportCodes.daml`

ten times each. For both stages time and memory consumption were measured and maximum, minimum, as well as average measurements were recorded. Subsequently the machine was rebooted after each test case. All diagrams in this report show the average of three measurement turns as described above.

The benchmarking tests were conducted on a Windows XP Workstation (3.0 GHz Intel Pentium 3 Processor, 1 GB physical RAM). KAON2, OWLIM, and Pellet were run in the same Java virtual machine (JVM) as the benchmarking application itself. RacerPro was running in a separate process and was connected using JRacer[3]. For all systems the JVM was set to initial and maximum heap size of 700 MB.

3.1 Starting Point: Existing ABox Benchmarks

Figure 1 shows the time needed to load different ontologies from LUBM. While RacerPro shows the worst performance and Pellet not being able to load the largest ontology, KAON2 turned out to be the fastest system directly followed by OWLIM. These two systems show a linear relationship between ontology size and load time.

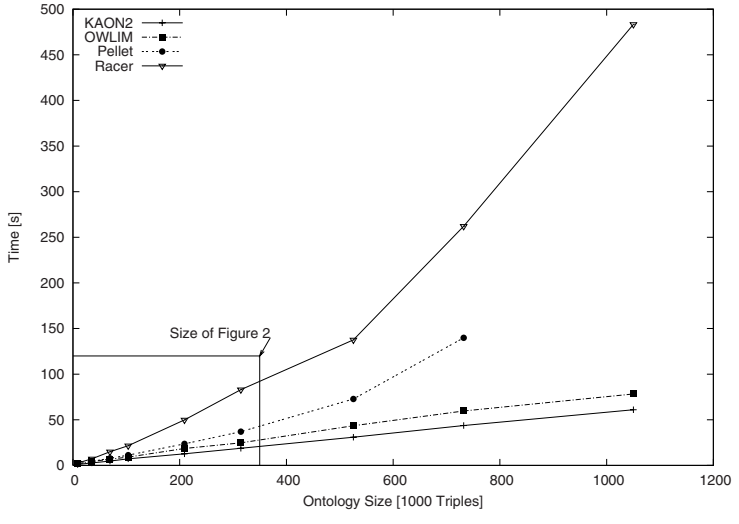

Fig. 1. Comparing LUBM load times for KAON2, Pellet, OWLIM, and RacerPro

We compared these results with the Semintec Benchmark which is based on a benchmark suggested by [20][4]. The Semintec ontology[5] consists of an even simpler TBox that even does not contain existential quantifiers or disjunctions.

[3] `http://www.racer-systems.com/products/download/nativelibraries.phtml`

[4] We only used the second of the two queries suggested in [20] since the concept `Person` (referenced in query one) is not present in the ontology.

[5] The Semintec ontology was originally created by the Semintec project: `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

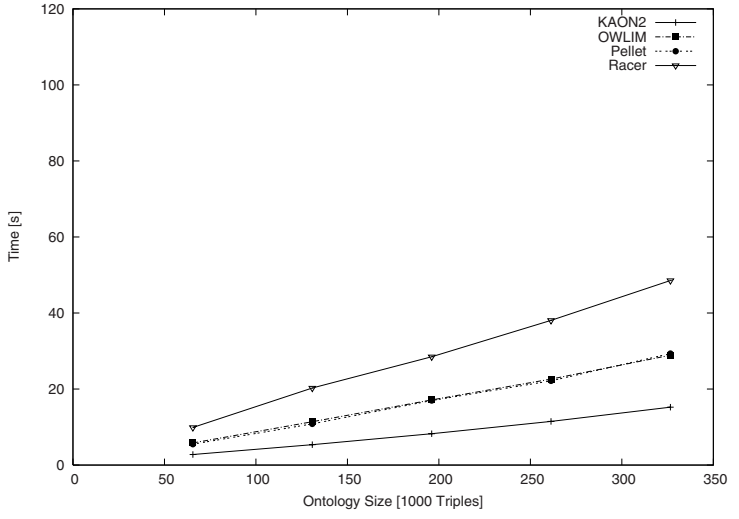

Fig. 2. Semintec Benchmark load times for KAON2, RacerPro, and OWLIM

Again we measured a somewhat linear relationship between the size of the ontology and the loading time for the named systems (cf. Figure 2). But we also noticed that RacerPro is only marginally slower than the other systems, in contrast to the LUBM. It seems that the lower expressivity also reduces the complexity of index creation during preprocessing.

3.2 Implicit Knowledge - A Stumbling Block?

The results from LUBM and the Semintec Benchmark were in general unspectacular, even though the small difference between the benchmarks could not be explained definitely. Thus we designed a Benchmark consisting of a very simple TBox for the next tests.

The TBox of the so called "Exquant Benchmark" consists of a transitive property and a concept defined as existential quantification upon this transitive property (`someValueFrom` restriction). The ABox consists of a chain of individuals related via the transitive property. This individual chain is of different length for every ontology in the benchmark, where 100.000 instances marks the maximum length. The query collects the extension of the restriction. The layout of this benchmark reflects one aspect of the social network ontology (part of our application scenario), which heavily uses transitive properties.

Suddenly, the picture changes. OWLIM, performing very well for LUBM and Semintec, is unable to load an ontology consisting of a chain of 1.000 individuals linked by a transitive property (all tests interrupted after 1 hour). In contrast RacerPro and KAON2 never needed longer than 3.5 seconds. Obviously OWLIM's total forward chaining and materialization approach to compute all

implicit knowledge on loading the ontology causes this performance deficit[6]. In the Exquant Benchmark the amount of implicit knowledge grows quadratic with the size of the ontology.

This also influences KAON2. Even though the system performs slightly better than RacerPro on loading the ontology, KAON2 was unable to answer the mentioned query, even for some of the smallest ontologies (a 500 element individual chain) within the given time limit of 10 minutes.

3.3 Influence of Serialization

Our next benchmark (the List Benchmark) consists of head|tail lists modeled in OWL. The biggest ontology contains a list of 100.000 elements. All ontologies in this benchmark are present in two different OWL serializations. One serialization follows a "flat" approach in the sense that all list elements are defined one after the other, referencing their respective tail. In the alternative "deeply nested" serialization, list elements are defined at the place where they are used.

An interesting result, when processing the list benchmark was that RacerPro is sensitive to the OWL serialization of the ontology loaded. We found that RacerPro easily loads the flat serialization of the List Benchmark, while the system fails to load deeply nested serializations with more then 6.400 list elements.

This emphasizes that reasoners should not be reduced to the performance of their core reasoning component when selecting a system for an application. Weaknesses might appear at unexpected places.

3.4 TBox Complexity

We are convinced that an ABox benchmark can not be conducted without scaling the TBox size, too. Inevitably this will also increase TBox reasoning complexity which again might influence ABox reasoning performance. Thus as a first test set we created the Unions Benchmark which checks the influence of TBox complexity on ABox reasoning. The benchmark primarily consists of a set of ontologies with gradually increasing TBox complexity. For every TBox, a set of ontologies with a growing number of ABox individuals is created.

The different TBoxes all consist of a concept hierarchy tree, in which every concept (except for leaf concepts) is defined as a union of further concepts modeled the same way. The TBox size is controlled by the number of concepts per union and the depth of the hierarchy tree. We then scale the size of the ABoxes by instantiating different amounts of individuals per concept. The query collects the extension of the root concept of the concept hierarchy, representing the superset of all ABox individuals.

Once again different reasoning techniques show different performance characteristics in this benchmark. While RacerPro's performance when querying the Unions Benchmark seems to solely depend on the size of the ABox, KAON2 mainly depends on the complexity of the TBox. Figures 3 and 4 depict these findings (pls. observe the direction of the level curves on the base of the graphs).

[6] Reportedly OWLIM v2.8.5 will feature optimized handling of transitive properties.

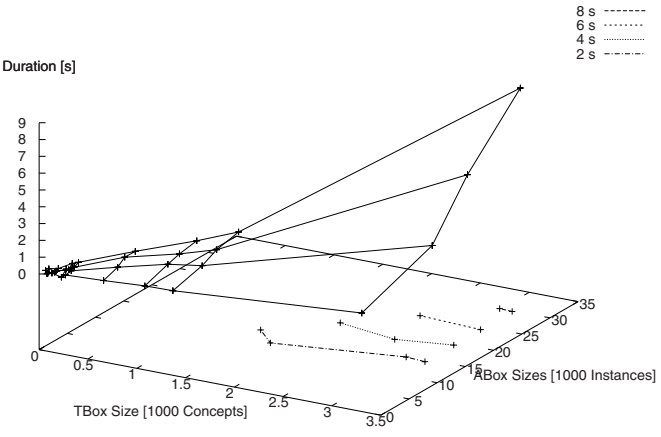

Fig. 3. RacerPro's query times for Unions Benchmark

In an additional test we introduced TBox Axioms irrelevant for the actual reasoning task. We suspected that some reasoners might switch off some optimizations in the presence of TBox axioms of higher complexity. Initial tests suggest that we were too censoriously regarding this assumption as we could not measure any differences.

3.5 Query Repetition and Query Specialization

We introduced the Query Specializing Benchmark to determine whether reasoners do profit from previously calculated results or not. If so, the executing of a specialization of a previous query would perform better than the execution of the specialized query alone.

We defined a set of five queries in selecting publications and their respective authors from the LUBM ontologies. Thereby, we restricted the possible authors from `Person` over `Faculty`, and `Professor` to `FullProfessor`. The last query additionally restricted the possible authors to `FullProfessors` working for a given department. The queries were processed against a "3 universities" ontology from most specific to most general and vice versa.

Unfortunately, we were not able to measure any significant speed up in comparison to the independent execution of the queries. Curious enough we were even unable to measure effects for RacerPro using its "query repository" [22] which is designed to make use of previously calculated answers.

Even if the same query is repeated several times, the query times do not necessarily decrease after the first execution. Considering all measurements we were not able to detect a significant speed up for KAON2 and only minor improvements (under 15%) for Pellet and RacerPro without query repository. OWLIM saved approximately one third of the initial query time, while the biggest speed

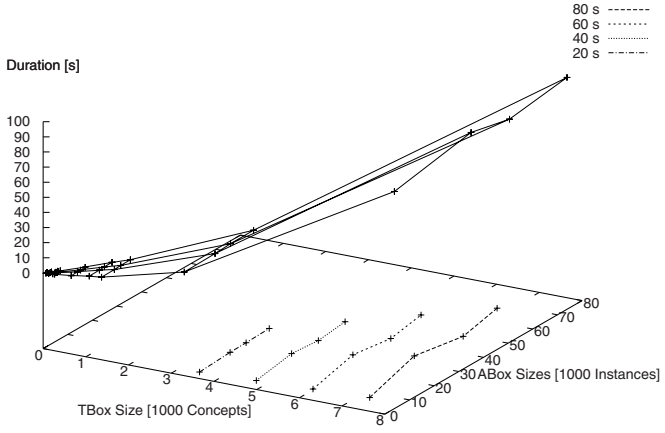

Fig. 4. KAON2's query times for Unions Benchmark

up was measured for RacerPro with activated query repository. Only in this configuration, repeated query executions in average were seven times faster compared to the first execution.

3.6 Dynamic Behavior

Existing performance results of DL reasoners are often limited to the classification of static ontologies. However, in the case of frequent updates (a KB submission, discarding, and re-submission cycle) the communication overhead introduced on loading the ontology can easily dominate the overall performance. In this respect, the delay caused by ontology-based inferencing easily becomes a major obstacle for its use in context-aware applications [23]. One approach to realize high-level situational reasoning for this type of application is to apply dynamic assertional classification of situation descriptions represented as concrete ABox individuals. Each situation individual is assembled of a set of ABox entities representing qualitative context information such as the location (e.g., office), the time (e.g., afternoon) and the persons in proximity (e.g., friends). Finally, the direct subsuming concepts of the situation individual determine the user's abstract situation. The whole process of determining the situation of a user (including the gathering and transformation of the relevant context data) is limited to about 2 seconds per classification. Retraction improves the performance for this type of application drastically, since only a small fraction of the ontology changes between two requests.

The standard DL interface DIG 1.1 [6] does not support the removal of specific axioms, making it necessary to re-submit the complete ontology for each request. As active members of the informal DIG 2.0 working group[7] we therefore propose

[7] `http://dig.cs.manchester.ac.uk/`

a modular extension to the interface that supports incremental reasoning and retraction [24]. Unfortunately, current reasoners only provide some kind of batch-oriented reasoning procedure. A notable exception is RacerPro, which offers low-level retraction support for most of its statements.

We compared different retraction strategies implemented in Racer. Reloading of ontologies from a local Web server can be accelerated by either loading from a image file (up to 3 times faster) or by cloning an ontology in memory (up to 70 times faster). For small ABoxes, cloning the ontology outperformed even the retraction of single axioms with forget statements (usually 80 times faster). However, it turned out that the fastest strategy was to keep situation individuals up to a certain number (about 20 in our case) within the ABox before cloning a fresh pre-loaded ABox.[8] Due to the lack of incremental classification algorithms, RacerPro still initiates a complete reclassification after each change in the ontology. Initial empirical results from [7], performed with an experimental version of Pellet, indicate that such algorithms for $\mathcal{SHOIN}(\mathcal{D})$ can be quite effective.

Without retraction support, the time needed to compute simple reasoning problems, is easily dominated by the communication overhead caused by the reasoner interface. For example, accessing RacerPro via its native API using TCP is about 1,5 times faster then via HTTP/DIG and even 2 times faster than the access realized with the triple-oriented framework Jena2 [25]. The best performance can be achieved by using the Pellet reasoner running in the same Java virtual machine as the application itself, this way without the need for any external communication.

Another problematic issue we observed was that some reasoners tend to allocate more and more memory over time. This leads to a considerable decrease in performance and makes it necessary to restart the reasoning component after a certain amount of transactions.

3.7 Completeness Versus Performance

Reasoning with OWL Lite as well as OWL DL is known to be of high worst-case complexity. By using the "right" combination of costly expressions, one can intentionally or incidentally create even a very small ontology whose complexity will make practical reasoning impossible. Therefore, in case of taking the whole vision of the Semantic Web literally as the domain for reasoning-aware applications, one obviously has to give up soundness and completeness [26]. However, besides some preliminary empirical evaluation [27], there are currently no attempts to reason with all ontologies found on the Web in parallel. Instead, when assuming the currently more realistic application range in which applications need to reason about information represented via distributed ontologies, soundness and completeness typically do matter. It seems very unlikely that users of large scale ontologies in the context of industrial or scientific research such as SWEET or GO, or defense critical approaches such as the "Profiles in Terror"

[8] Keeping individuals and axioms in the ABox is only possible if they do not influence later classifications.

ontology will accept incomplete reasoning results. Note that in the presence of full negation, as in OWL, one can not really distinguish between completeness and correctness anymore. Because the answers you miss due to incompleteness will be your incorrect answers of the complementary problem.

As a consequence we tested our systems with help of an empirical evaluation using spot tests which are intentionally designed to be hard to solve but small in size. They try to meter the correctness of the reasoning engines with respect to inference problems of selected language features.[9] Surprisingly, only RacerPro and KAON2 were able to solve those tests which lay within the language fragment (above $\mathcal{ALC}$) they claim to support. Others such as Pellet and FaCT++ even failed on some OWL Lite test cases (not to mention OWLIM and related systems). Besides this semantical errors we also found a couple of parsing problems. For instance, all of the systems failed to parse either an empty intersection, union, or enumeration via XML/RDF or DIG 1.1. We also experienced that there is an unpredictable scatter of runtime from case to case even within one system implementation. Actually we discovered random runtime behavior for Pellet for one test case ranging from less than a second up to effectively non-termination. Finally, an expressive all-embracing test case with less than 50 classes and individuals overextended almost all systems.

The discovered failures have been communicated to the system developers and a more detailed description of our test suite can be found at [29].

In addition, we found out that the given answer sets of UOBM are wrong in the DL part of the benchmark suite. Their approach of importing all statements into a RDBMS and manually build SQL queries for answer set computing failed for query 11 with five universities for example. The presumably correct number of answers is 6230 (as opposed to the official 6225) and was computed by Pellet. At least the additional retrieved individuals from Pellet represent correct answers. This can easily be seen by manually collecting the information which makes them a legal candidate. As far as we see the official result set does not take into account that `isHeadOf` is an inverse functional property.

4 Requirements for a Comprehensive ABox Benchmark

In the above sections we demonstrated the impact of some important influencing factors neglected by today's standard ABox benchmarks. This weakness renders the named benchmarks useless when choosing a reasoner for a real-world application. We thus suggest to build future benchmarks along the lines of the following requirements.

R1 Separate measurements should be taken for each processing stage (loading and querying) as described in Section 3.

R2 The benchmark should investigate query performance when processing a set of ontologies with gradually growing ABoxes while size and complexity of

[9] Very similar to the system evaluation of [28] and the system comparisons conducted at various DL workshops.

the TBox remains constant. It must thereby be ensured that the ABox can't be split into disjoint and unconnected parts.

R3 The benchmark should also pinpoint the influence of TBox complexity on ABox reasoning. Thus TBox complexity should gradually be increased. In one setting this increase in complexity should influence the ABox reasoning task while in a separate setting TBox axioms which are unrelated to the actual benchmark should be added. The second setting is to trick the reasoner into switching off optimizations even if this would not have been necessary for the actual reasoning task.

R4 Include benchmarks, that comprise TBoxes modeled in a way such that adding explicit knowledge to the ABox also adds large quantities of implicit knowledge (e.g. transitive properties). This is to reveal the possibly negative influences of materialization approaches or maintenance of index structures.

R5 OWL allows for different serializations of the same ontology. The benchmark should check the influence of different serializations on the process of loading these ontologies. A well implemented reasoner should be agnostic to such differences.

R6 A reasoner with well implemented query caching should answer a repetition, a specialization, or a sub query of a previous query almost instantly. Thus tests should be included which disclose the reasoners capabilities with respect to query caching.

R7 Most reasoners support different interfaces, like a proprietary API and a DIG interface. Since these interfaces might exhibit different performance the benchmark should compare loading and processing of ontologies through the varying interfaces. Clearly results from this benchmark can be disregarded if only very time consuming reasoning tasks are triggered. In such cases the communication overhead is negligible.

R8 Real world applications will be subject of constant change. These changes will appear most frequently in the ABox. Thus additional benchmarks should be available measuring the performance of ABox modifications like addition or retraction of axioms and the time required for subsequent reasoning tasks.

A future comprehensive ABox benchmark should consist of a set of specialized benchmarks tackling a variety of different requirements. Though, we won't suggest to define a procedure to reduce the various results of the different benchmarks to a single metric (as done in [2]). Because, we do not believe that a single score would be of any help when selecting a reasoner for a particular application scenario. For instance, consider the case where two reasoners claim to support the same expressivity but one of them is not sound/incomplete. Then, strictly speaking, they are not comparable at all. Therefore, as a kind of meta requirement, comparisons should carefully interpret all measured results with respect to the different underlying approaches and their theoretical properties.

From a practical point of view, it is advisable to analyze the specific requirement of the planned application and then choose the relevant benchmarks for comparison of potential reasoners. In this respect a set of special purpose benchmarks will be of great help. As a starting point we compiled Table 1, that lists

Table 1. Requirements covered by the benchmarks presented in this paper

Benchmark	Description	Meets Requirements
LUBM	The original Lehigh University benchmark	R2
UOBM	Extended LUBM which introduces an OWL Lite and an OWL DL Version of the benchmark	R2, R3 partially
Semintec	Based on a real-word TBox, modeling the financial domain. ABox size is increased in five steps.	R2
List	Synthetic ontology modeling a head|tail list in OWL. Amount of implicit knowledge rises exponentially with the number of list elements.	R2, R4, R5
Exquant	Another synthetic ontology heavily using transitive property instances.	R2, R4
Unions	Benchmark that increases ABox size as well as TBox complexity	R2, R4, R5, R3 partially
Query Specializing	Based on LUBM. Consists of increasingly specialized queries. Checks for query caching capabilities.	R2, R6

the benchmarks presented in this paper together with the covered requirements (requirements R0 and R1 are not mentioned there as they are independent of the concrete benchmark).

5 Summary

We showed that today's ABox benchmarks fall short on providing comprehensive and meaningful information in order to support users in selecting a reasoner for real world applications. We highlighted and discussed some benchmarking results gained from well known as well as newly created benchmarks. These benchmarks cover traditional aspects like ABox size but also measure influences due to ontology serialization, TBox complexity, query caching, and dynamic ontology changes. The results clearly show that there is still no single benchmark suite which covers all of the issues above and that there is no reasoner able to deal with large and complex ABoxes in a robust manner. As a consequence we suggest a set of general benchmarking requirements which will be helpful when designing future OWL reasoner benchmarking suites.

References

1. Luther, M., Böhm, S., Wagner, M., Koolwaaij, J.: Enhanced Presence Tracking for Mobile Applications. In: Proc. of 4th Int. Semantic Web Conference (ISWC'05), Galway, Ireland. Volume 3729., Galway, Ireland, Springer (2005)
2. Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: Proc. of the 3rd Int. Semantic Web Conference (ISWC'04), Hiroshima, Japan (2004) 274–288

3. Möller, R., Haarslev, V., Wessel, M.: On the Scalability of Description Logic Instance Retrieval. In: Proc. of the 29th German Conf. on Artificial Intelligence. LNAI, Bremen, Germany, Springer (2006) 171–184
4. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a Complete OWL Ontology Benchmark. In: Proc. of the 3rd European Semantic Web Conference (ESWC'06). Volume 4011 of LNCS., Budva, Montenegro, Springer (2006) 125–139
5. Brachman, R., Levesque, H.: The Tractability of Subsumption in Frame-based Description Languages. In: Proc. of the 4th Nat. Conference on Artificial Intelligence (AAAI'84). (1984) 34–37
6. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In: Proc. of the Int. Workshop on Description Logics (DL'03). Volume 81 of CEUR., Rome, Italy (2003)
7. Halaschek-Wiener, C., Parsia, B., Sirin, E., Kalyanpur, A.: Description Logic Reasoning for Dynamic ABoxes. In: Proc. of the Int. Workshop on Description Logics (DL'05), Edinburgh, Scotland. Volume 147 of CEUR. (2006)
8. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL DL Reasoner. Journal of Web Semantics (2006) To Appear.
9. Haarslev, V., Möller, R.: Racer: A core inference engine for the Semantic Web Ontology Language (OWL). In: Proc. of the 2nd Int. Workshop on Evaluation of Ontology-based Tools. (2003) 27–36
10. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Proc. of the Int. Joint Conference on Automated Reasoning (IJCAR'06). (2006) To Appear.
11. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The Summary Abox: Cutting Ontologies Down to Size. In: Proc. of the 5th Int. Semantic Web Conference (ISWC'06), Athens, GA, USA, Springer Verlag (2006) 343–356
12. Motik, B., Studer, R.: KAON2 – A Scalable Reasoning Tool for the Semantic Web. In: Proceedings of the 2nd European Semantic Web Conference (ESWC'05), Heraklion, Greece (2005)
13. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM — a Pragmatic Semantic Repository for OWL. In: Proc. of the Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'05), New York City, USA, Springer (2005) 182–192
14. Acciarri, A., Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), Pittsburgh, USA, The MIT Press (2005) 1670–1671
15. Zhou, J., Ma, L., Liu, Q., Zhang, L., Yu, Y., Pan, Y.: Minerva: A Scalable OWL Ontology Storage and Inference System. In: Proc. of 1st Asian Semantic Web Conference (ASWC 2006), Beijing, China, Springer Verlag (2006) 429–443
16. Bechhofer, S., Horrocks, I., Turi, D.: The OWL Instance Store: System Description. In: Proc. of the 20th International Conference on Automated Deduction (CADE 2005), Tallinn, Estonia, Springer Verlag (2005) 177–181
17. Chen, C., Haarslev, V., Wang, J.: LAS: Extending Racer by a Large Abox Store. In: Proc. of the Int. Workshop on Description Logics (DL'05), Edinburgh, Scotland, UK (2005) 200–2007
18. Gardiner, T., Horrocks, I., Tsarkov, D.: Automated Benchmarking of Description Logic Reasoners. In: Proc. of the Int. Workshop on Description Logics (DL'06), Lake District, UK. Volume 189 of CEUR., Lake District, UK (2006) 167–174
19. Haarslev, V., Möller, R., Wessel, M.: Querying the Semantic Web with Racer + nRQL. In: Proc. of the 3rd Int. Workshop on Applications of Description Logics (ADL'04). CEUR, Ulm, Germany (2004)

20. Motik, B., Sattler, U.: A Comparison of Techniques for Querying Large Description Logic ABoxes. In: Proc. of the 13th Int. Conf. on Logic Programming Artificial Intelligence and Reasoning (LPAR'06). Volume 4246 of LNCS., Phnom Penh, Cambodia, Springer (2006)
21. Fokoue, A., Kershenbaum, A., L., M., Schonberg, E., Srinivas, K.: The Summary Abox: Cutting Ontologies Down to Size. Technical Report TR-404, IBM Research – Intelligent Application Analysis, Hawthornem, NY (2006)
22. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In: Proc. of the Int. Workshop on Description Logics (DL'05), Edinburgh, Scotland, UK (2005) 84–95
23. Luther, M., Fukazawa, Y., Souville, B., Fujii, K., Naganuma, T., Wagner, M., Kurakake, S.: Classification-based Situational Reasoning for Task-oriented Mobile Service Recommendation. In: Proc. of the ECAI'06 Workshop on Contexts and Ontologies. (2006)
24. Bechhofer, S., Liebig, T., Luther, M., Noppens, O., Patel-Schneider, P., Suntisrivaraporn, B., Turhan, A., Weithöner, T.: DIG 2.0 – Towards a Flexible Interface for Description Logic Reasoners. In: Proc. of the OWL Experiences and Directions Workshop (OWLED'06) at the ISWC'06. (2006)
25. Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. Technical Report HPL-2003-146, HP Labs (2004)
26. van Harmelen, F.: How the Semantic Web will change KR: challenges and opportunities for a new research agenda. The Knowledge Engineering Review **17** (2002) 93–96
27. Guo, Y., Qasem, A., Heflin, J.: Large Scale Knowledge Base Systems: An Empirical Evaluation Perspective. In: Proc. of the 21st National Conf. on Artificial Intelligence (AAAI 2006), Boston, USA (2006) to appear.
28. Heinsohn, J., Kudenko, D., Nebel, B., Profitlich, H.J.: An Empirical Analysis of Terminological Representation Systems. Technical Report RR-92-16, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany (1992)
29. Liebig, T.: Reasoning with OWL – System Support and Insights –. Technical Report TR-2006-04, Ulm University, Ulm, Germany (2006)