# What's Wrong with OWL Benchmarks?

Timo Weithöner[1], Thorsten Liebig[1], Marko Luther[2], and Sebastian Böhm[2]

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

**Abstract.** This paper is motivated by experiences in course of developing an ontology-based application within a real-world setting. When comparing these experiences with results of well-known benchmarks we found out that current benchmarks are not well suited to provide helpful hints for application developers who seek for a reasoning system matching typical real-world needs. This work aims at identifying requirements to make future benchmarks more useful for application developers.

## 1 On Benchmarking OWL Reasoners

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. On the other hand, recent implementations have shown significant progress with respect to the scalability of instance retrieval over ABoxes containing several hundred thousands of individuals. These observations cannot be seen as separate issues. Quite the contrary, they are opposing landmarks on the map of complexity results whose topology is characterized by a trade-off between expressivity and efficiency. This trade-off typically comes as a relatively sharp border between tractable and effectively un-tractable ontologies, the so called *computational cliff*, and is caused by an increase in language expressiveness [1]. Although a worst-case result, in practice this behavior is often observed with tableaux based systems when adding language constructs that prohibit the use of highly efficient optimization techniques (i. e. inverse roles, nominals, etc.). Eventually, a multiplier to this effect is the size of the ABox.

Having knowledge about the rough landscape of practically tractable ontologies as well as reasoning approaches for particular query requirements is of fundamental importance when building real-world applications. By real-world we mean ontology-based applications with an expressivity beyond $\mathcal{ALC}$, containing thousands of individuals and an inference response time of less than a second, even in a dynamical setting. 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 [2] clearly have shown that the quality of this application significantly depends on the availability of

scalable reasoning services able to deal with dynamic data. Within this application user situations are computed by applying Description Logic reasoning to classify qualitative context elements gathered from various sources and represented as OWL individuals [3]. Since changes in the user's environment occur frequently the corresponding ABoxes have to be updated rather often.

Unfortunately, none of the current benchmarks or system comparisons either draw a clear picture of effectively tractable language fragments or ABox sizes, give insights into pros and cons of different reasoning approaches, or consider technical issues such as serialization, interfaces, or ABox updates.

For instance, many benchmarks consist of synthetical generated and sparsely interrelated data using inexpressive ontology languages such as the widely used Lehigh University Benchmark (LUBM) [4]. In case of the LUBM an incomplete query answering procedure exploiting told ABox information from a classified TBox is sufficient to answer most of the given queries correctly. Obviously, this cannot be considered as a meaningful benchmark and consequently has let to exceptional performance for inherently incomplete reasoning systems.

On the other hand, a demonstration of the steep slope of the computational cliff is the University Ontology Benchmark (UOBM) [5], which is a direct extension of the LUBM. The UOBM 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 set of given UOBM queries. Not surprisingly, it turned out that incomplete systems such as OWLIM, DLDB, or Minerva now can only answer a fraction even of the OWL Lite queries completely. Only one theoretically sound and complete approach, namely Pellet [6], was able to handle about a tenth of the number of individuals compared to the LUBM within our time limit. All other systems failed either with a timeout or due to the lack of memory.

Furthermore, discussing 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 the trade-off when using a rule-based materialization approach such as in OWLIM [7], a Datalog-driven engine such as KAON2 [8], or tableau-based reasoners such as Pellet, RacerPro [9] or FaCT++ [10]. To give an example, a materialization approach has to exhaustively compute and explicitly store all implicit information of an ontology typically at load-time. On the contrary, traditional tableau-based systems will explicitly compute such information only on demand. A Datalog transformation approach, on the other hand, is known to perform worse in the presence of numbers greater than two in cardinality restrictions. Therefore, a serious benchmark has to discuss its results with respect to different approaches in order to provide helpful hints to users.

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 [11]) 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. [12], 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. In fact, only RacerPro and KAON2 passed all tests of the OWL fragment they claim to support completely.

The purpose of this paper is to identify weaknesses of current OWL benchmarks and system evaluations. In the following section, we analyze existing benchmark suites, discuss corresponding results and compare them with our own tests. As a result we compiled a collection of requirements (Section 3) to make future benchmarks more useful for application developers. Section 4 summarizes the findings presented in this report.

## 2 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. 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.

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[3] or Airport Codes Ontology[4]) which are used for benchmarking can be found in [13].

Nowadays, the Lehigh University Benchmark (LUBM) is the de facto standard when it comes to reasoning with large ontologies [14,15,6,16,17]. 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

[3] `http://archive.godatabase.org/`
[4] `http://www.daml.ri.cmu.edu/ont/AirportCodes.daml`

for OWLIM and KAON2 from Sections 2.1 and 2.2). We therefore experimented 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. 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)[5], Pellet 1.3, OWLIM 2.8.3, and RacerPro 1.9.0. 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.

**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 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 aggregate the numbers achieved from 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 where 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[6]. For all systems the JVM was set to initial and maximum heap size of 700 MB.

### 2.1 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 cannot load the largest ontology, KAON2 is the fastest system directly followed by OWLIM. These two systems show a linear relationship between ontology size and load time.

[5] New release available since 28-07-2006; another followed 29-08-2006

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

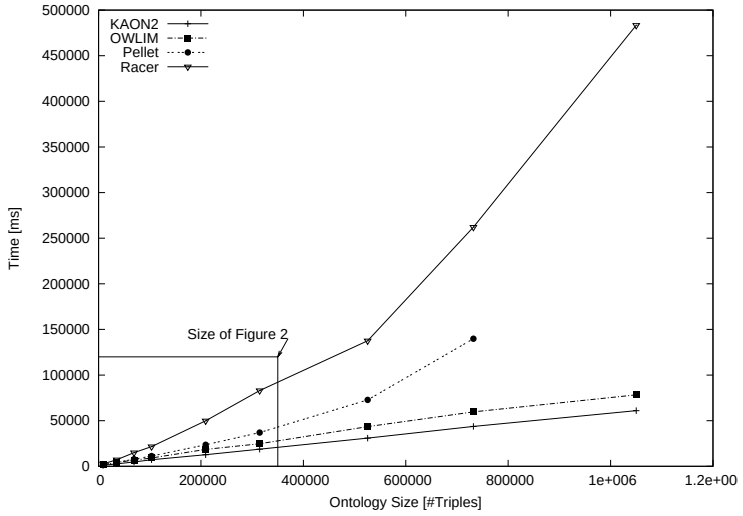


**Fig. 1.** Comparing LUBM load times for KAON2, Pellet, OWLIM, and RacerPro

We compared these results with the Semintec Benchmark which is based on a benchmark suggested by [16][7]. The Semintec ontology[8] consists of an even simpler TBox that even does not contain existential quantifiers or disjunctions.

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 seams that the lower expressivity also reduces the complexity of index creation during preprocessing.

### 2.2 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.

[7] We only used the second of the two queries suggested in [16] since the concept `Person` (referenced in query one) is not present in the ontology.

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

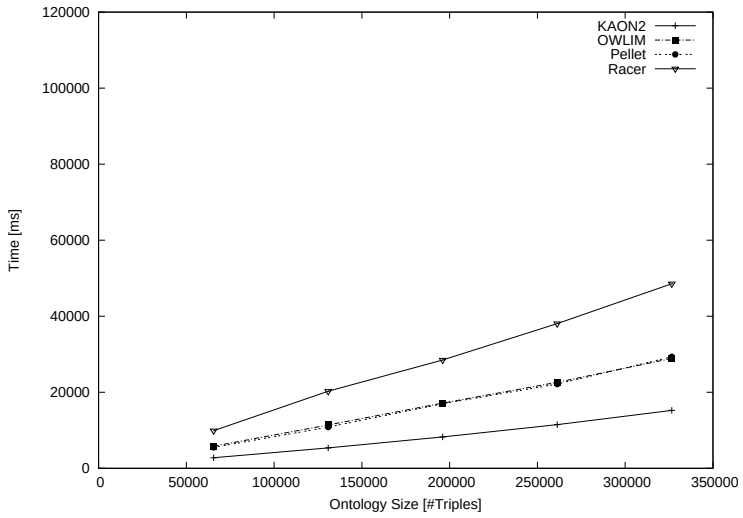


**Fig. 2.** Semintec Benchmark load times for KAON2, RacerPro, and OWLIM

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. In the Exquant Benchmark the amount of implicit knowledge grows exponentially 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.

### 2.3 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.

### 2.4 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 seams to solely depend on the complexity of the TBox, KAON2 mainly depends on the size of the ABox. Figures 3 and 4 depict these findings (pls. observe the direction of the level curves on the base of the graphs).

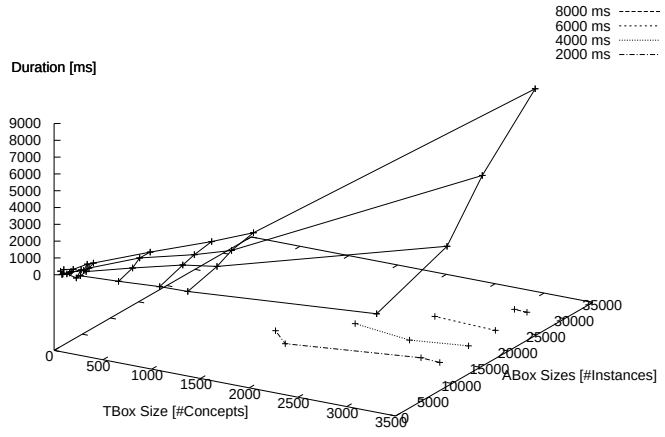


**Fig. 3.** RacerPro's query times for Unions Benchmark

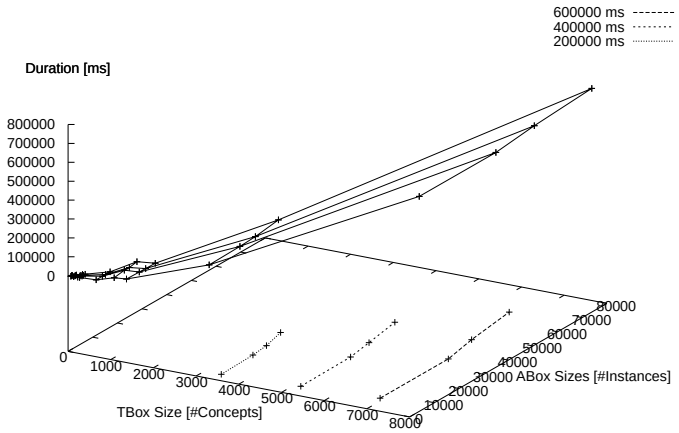


**Fig. 4.** KAON2's query times for Unions Benchmark

### 2.5 Query Repetition and Query Specializing

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 `FullProfessor`s 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" [18] 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 up was measured for RacerPro with actived query repository. Only in this configuration, succeeding query executions in average were seven times faster compared to the first execution.

### 2.6 Bluffing Optimizers

Since lots of work has already been conducted in the field of reasoner optimizations, we were interested in the quality of the resulting optimization algorithms. Especially we suspected that reasoners could easily be tricked into switching off certain optimizations by adding TBox axioms of higher complexity – even if these axioms were not relevant for the actual reasoning task. Initial tests suggest that we were too censoriously at this point as we could not measure differences so far.

### 2.7 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 [3]. 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 [11] 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[9] we therefore propose a modular extension to the interface that supports incremental reasoning and retraction [19]. 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

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

a fresh pre-loaded ABox.[10] Due to the lack of incremental classification algorithms, RacerPro still initiates a complete reclassification after each change in the ontology. Initial empirical results from [12], 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 [20]. 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.

### 2.8 Completeness versus Performance

In case of taking the whole vision of the Semantic Web literally as the domain for reasoning-aware applications one obviously has give up soundness and completeness [21]. However, besides some preliminary empirical evaluation [22], there are currently no attempts to reason with all ontologies found on the web in parallel. However, when assuming the currently more realistic application range in which applications need to reason about information represented as distributed ontologies, soundness and completeness typically do matter. It seems very unlikely that users of large scale ontologies in the context of scientific research such as SWEET or GO, or defense critical approaches such as the "Profiles in Terror" ontology will be comfortable with incomplete reasoning results.

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.[11] Surprisingly, only RacerPro and KAON2 were able to solve those tests which lay within the language fragment 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).

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

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

[11] Very similar to the system evaluation of [23] and the system comparisons conducted at various DL workshops.

answers is 6230 (as opposed to the official 6225) and was computed by Pellet. At least the additional retrieved individuals from Pellet are correct answers which can easily be seen by manually collecting the information which make them a legal candiate. They differ from the officially computed individuals in that they require to take into account that `isHeadOf` is an inverse functional property.

## 3 Requirements for a Comprehensive ABox Benchmark

In the above sections we demonstrated the impacts 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.

**R0** Despite the fact that all reasoning results from any benchmark should always be checked for soundness and completeness, we suggest to start every ABox benchmarking session with a rigorous check of the overall reasoning capabilities of the involved reasoners. This should be observed especially when different systems are compared in the following benchmarking steps.

**R1** Separate measurements should be taken for each processing stage (loading and querying) as described in Section 2.

**R2** The benchmark should investigate query performance when processing a set of ontologies with gradually growing ABoxes while size and complexity of 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. A well implemented reasoner should thus continue with optimizations whereas other systems might show a significant decrease in performance.

**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 will most likely consists of a set of different specialized benchmarks appropriate for the different requirements. Though, we won't suggest to define algorithms to reduce the various results of the different benchmarks to a single metric. Furthermore, we do not believe that a single score would be of any particular help when selecting a reasoner for a given application scenario. This is especially true as there are no two reasoners truly supporting the same expressivity. 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 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).

**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 |
| **Bluffing Optimizers** | Checks if TBox axioms irrelevant for the actual reasoning task slow down the reasoning process. | R3 partially |

## 4 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 achieved 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 as well as 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. 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
2. Luther, M., Böhm, S., Wagner, M., Koolwaaij, J.: Enhanced Presence Tracking for Mobile Applications. In Gil, Y., Motta, E., Benjamins, V., Musen, M., eds.: Proc. of 4th Int. Semantic Web Conference (ISWC'05), Galway, Ireland. Volume 3729., Galway, Ireland, Springer (2005)
3. 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)
4. 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
5. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a Complete OWL Ontology Benchmark. In Sure, Y., Domingue, J., eds.: Proc. of the 3rd European Semantic Web Conference (ESWC'06). Volume 4011 of LNCS., Budva, Montenegro, Springer (2006) 125–139
6. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL DL Reasoner. Journal of Web Semantics (2006) To Appear.
7. 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
8. 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)
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. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In Calvanese, D.e.a., ed.: Proc. of the Int. Workshop on Description Logics (DL'03). Volume 81 of CEUR., Rome, Italy (2003)
12. Halaschek-Wiener, C., Parsia, B., Sirin, E., Kalyanpur, A.: Description Logic Reasoning for Dynamic ABoxes. In Horrocks, I., Sattler, U., Wolter, F., eds.: Proc. of the Int. Workshop on Description Logics (DL'05), Edinburgh, Scotland. Volume 147 of CEUR. (2006)
13. Gardiner, T., Horrocks, I., Tsarkov, D.: Automated Benchmarking of Description Logic Reasoners. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. of the Int. Workshop on Description Logics (DL'06), Lake District, UK. Volume 189 of CEUR., Lake District, UK (2006) 167–174
14. Möller, R., Haarslev, V., Wessel, M.: On the Scalability of Description Logic Instance Retrieval. In Freksa, C., Kohlhase, M., eds.: Proc. of the 29th German Conf. on Artificial Intelligence. LNAI, Bremen, Germany, Springer (2006) 171–184
15. 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)
16. Motik, B., Sattler, U.: A Comparison of Techniques for Querying Large Description Logic ABoxes. In Hermann, M., Voronkov, A., eds.: Proc. of the 13th Int. Conf. on Logic Programming Artificial Intelligence and Reasoning (LPAR'06). LNCS, Phnom Penh, Cambodia, Springer (2006) To Appear.
17. 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)
18. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In Horrocks, I., Sattler, U., Wolter, F., eds.: Description Logics. Volume 147 of CEUR Workshop Proceedings., CEUR-WS.org (2005)
19. 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) To Appear.
20. 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)
21. 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
22. 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.
23. 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)