Analysis of Test-Results on Individual Test Ontologies

Deliverable TONES-D23

G. De Giacomo[2], E. Franconi, B. Cuenca Grau[3], V. Haarslev[5], A. Kaplunova[5], A. Kaya[5], D. Lembo[2], C. Lutz[4], M. Miličić[4], R. Möller[5], U. Sattler[3], B. Sertkaya[4], B. Suntisrivaraporn[4], A.-Y. Turhan[4], S. Wandelt[5], M. Wessel[5]

[1] Free University of Bozen-Bolzano
[2] Università di Roma "La Sapienza"
[3] The University of Manchester
[4] Technische Universität Dresden
[5] Technische Universität Hamburg-Harburg

Project:	FP6-7603 – Thinking ONtologiES (TONES)
Workpackage:	WP7 – Experimentation and Testing of the Framework
Lead Participant:	TU Dresden
Reviewer:	Ralf Möller
Document Type:	Deliverable
Classification:	Public
Distribution:	TONES Consortium
Status:	Final
Document file:	D23_TestsOntologies.pdf
Version:	2.0
Date:	August 31, 2007
Number of pages:	73

Document Change Record		
Version	**Date**	**Reason for Change**
v.0.1	July 13, 2007	Outline
v.1.0	August 17, 2007	First complete version
v.2.0	August 31, 2007	Final version

Contents

1 Introduction

The purpose of this deliverable is to report about the results of testing and evaluating techniques that have been developed within the TONES project. We concentrate on techniques from the workpackages that, at the time of writing, are in their final phase. These are WP3 (Ontology Design and Maintenance) and WP4 (Ontology Accessing, Process and Usage). The techniques developed within WP5 and WP6, which have started significantly later, will be evalutated in deliverable D27. Thus, the techniques evaluated in this deliverable are those presented in deliverables D13 [BBC+07] and D18 [CGG+07a]. To achieve an efficient and practically useful implementation, the basic reasoning techniques have been enriched with a large number of implementation and optimization techniques, which are (partly) described in the reports accompanying the software deliverables D15 [CGF+07] and D21.

The techniques evaluation in this deliverable are implemented in eight different tools. Many of these tools are multi-purpuse, and they implement much more than a single technique and contribute to more than one workpackage. In particular, several of the tools used within this deliverable also contribute to the later workpackages WP5 and WP6. This will be discussed in detail in deliverable D27.

The tests carried out in this deliverable can be split into two groups. First, there are tests of so-called "standard" reasoning services, i.e., services that have a long tradition within the field of logic-based ontologies. Examples of such services include TBox classification on the intensional level of reasoning and ABox query answering on the extensional level. For these tests, we concentrate on evaluating optimization techniques with the goal of demonstrating *practical feasibility and scalability*, also in the context of ontologies and instance data of massive size. Second, there are tests of novel reasoning services that have been developed within the TONES project or not long before the start of this project. For these tests, the purpose of is to establish a *proof of concept* for usability of the reasoning techniques, i.e., to show that it is possible to implement the techniques so that they are sufficiently performant to be used on reasonable size data, and that their use is beneficial for ontology design and usage. In this latter case, it was usually not the aim to tease out the last bit of efficiency.

More specifically, the first group is comprised of tests of the following tools.

- RACER is a multi-purpose tool that mainly addresses standard reasoning tasks such as ontology classification and ABox query answering, and puts an emphasis on universality and efficiency. Our tests demonstrate two things. First, RACER allows to very efficiently classify ontologies formulated in expressive logics. It scales seamlessly to ontologies of large size and outperforms other state-of-the-art tools. Second, a suitable combination of the several optimization techniques developed within the TONES project allows to turn the tableau algorithm-based RACER into a very efficient and scalable tool for ABox query answering, while single optimizations fail to achive this goal.

- FACT++ is a tool for classifying ontologies formulated in expressive logics. In particular, FACT++ is currently the only reasoner that can handle the description

logic $\mathcal{SROIQ}$, which underlies the upcoming OWL 1.1 standard ontology language. Despite being able to handle this very high expressive power, FaCT++ is also a highly efficient and scalable reasoner.

- CEL. In contrast to Racer and FaCT++, the CEL tool focusses on classifying ontologies formulated in lightweight, relatively inexpressive logics. The purpose of this approach is to gain polytime reasoning and scalability to an even better degree than achieved by tools such as Racer and FaCT++. Our tests use massive-scale ontologies and demonstrate that the stated goal is very successfully achieved.

- QuOnto focusses, like CEL, on the processing of ontologies formulated in lightweight logics. Unlike CEL, QuOnto concentrates on ABox query answering rather than classification. The main idea of QuOnto is to achieve very good scalability by exploiting mature relational database technology for ontology reasoning. Indeed, our tests show that the performance of QuOnto is comparable to that of established database systems, and thus scales extremely well.

The second group consists of tests of the following tools.

- Sonic provides a variety of novel reasoning service such as the computation of least common subsumers and approximation concepts. Our tests show that even a not fully optimized implementation of the developed algorithm are usable on realistic ontologies from practical applications, that the runtimes are surprisingly good, and that the implemented approximation techniques typically do not lose significant information.

- We have also tested an approach to abduction in description logic. This tool is tailored towards a specific application, which is media interpretation, i.e., an ontology is used to interpret, generate, and improve annotations of a media file. Our tests show that ontology reasoning can significantly improve the quality of automatic scene interpretation and provides valuable feedback for the lower level image processing tools.

- InstExp is a tool that allows the interactive completion of an ontology by a domain expert. It relies on an underlying reasoner for classification such as Racer and FaCT++. Our tests show that a straightforward implementation is already very useful. However, they also show that incremental reasoning by the underlying standard reasoner is of great importance to improve performance, and suggests a number of extensions to the framework (such as allowing a domain expert to defer a question posed to him by InstExp).

- The query tool is meant to support a user in formulating a precise query even in the case of complete ignorance of the ontology vocabulary of the underlying information system holding the data. We perform a well-founded *usability* evaluation of the tool. The main goal of this experiment is to demonstrate the easy of use of the Query tool independently of the domain user experience.

2 Ontologies

Selecting a set of ontologies for testing and evaluating a reasoning technique is a difficult task. It is important to test multiple ontologies to demonstrate that systems are not tailored towards specific ontologies. It is necessary to use ontologies from practical applications because they provide the most realistic input; it is also necessary to use artificially generated ontologies and/or instance data because there is only a limited supply of ontologies from practical applications and those ontologies often have a very simple structure and may not be adequate to test a feature at hand. Some requirements for testing reasoning services on single ontologies are summarized in [WLLB06, WLL+07].

Within TONES, we have collected and made available a library of test ontologies, which includes mostly hand-crafted ontologies, but also a couple of artificially generated ones. Each ontology has been checked for expressivity (i.e., the logic in which it is formulated) and syntactic conformance, translated into DIG syntax (which is easy to work with for benchmarking purposes), and included (whenever possible) additional information such as the derived class hierarchy to support testing the correctness of reasoning systems. Currently, our library contains over 300 ontologies. Only 18% of the ontologies make full use of the expressivity of the DL $\mathcal{ALC}$ or its extensions, which confirms that the majority of real-world ontologies are not very complex. On the other hand, we are still left with a sufficient number of challenging examples.

Most of the ontologies that are used in this deliverable are from this library. However, there are a number of exceptions due to the fact that some of the ontologies that we have used for testing are confidential, and the TONES ontology library is being made publically available. More of the relevant ontologies have been introduced and described in Deliverable D14 [CGG+07b], and we refer the interested reader to that document for details. In the remainder of this section, we introduce additional ontologies that are used in this deliverable, but have not been mentioned in D14. We list them in alphabetical order together with a short description.

2.1 AEO

The Athletics Event Ontology[1] contains concepts, relations, axioms and rules that formally represent the domain of Athletics in the BOEMIE project[2]. The ontology is written in $\mathcal{SHIN}(\mathcal{D})$ and it has 179 concept names, 158 properties and contains 523 axioms.

2.2 FERMI

The FERMI[3] ontology was generated in the context of a project about formalization and experimentation on the retrieval of multimedia information. It is represented in $\mathcal{EL}$ and it contains both a TBox and an ABox. More information on FERMI can be found in Table 2.

[1] http://www.boemie.org/d3_2_domain_ontologies

[2] http://www.boemie.org

[3] http://www.dcs.gla.ac.uk/fermi/

2.3 FungalWeb

The FungalWeb[4] ontology is an outcome of a project using DL technology in the context of fungal genomics [SNBHB05, BSNS+06]. It is represented in $\mathcal{ALCH}(\mathcal{D})$ and it contains both a TBox and an ABox. More information on FungalWeb can be found in Table 2.

2.4 Galen

Galen[5] is concerned with the computerisation of clinical terminologies. It allows clinical information to be captured, represented, manipulated, and displayed in a more powerful way. The full Galen ontology is a large and complex medical ontology, represented in the Description Logic $\mathcal{SHIF}$, designed for supporting clinical information systems. In fact, it does not use the logical constructors: negation, disjunction and value restrictions, which makes large part of it (97.75%) expressible in the lightweight Description Logic $\mathcal{EL}^+$.

2.5 GO Daily

The Gene Ontology (GO)[6] project is a collaborative effort to address the need for consistent descriptions of gene products in different databases. The GO project has developed three structured controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. There are three separate aspects to this effort: first, the development and maintenance of the ontologies themselves; second, the annotation of gene products, which entails making associations between the ontologies and the genes and gene products in the collaborating databases; and third, development of tools that facilitate the creation, maintenance and use of ontologies. The ontology is represented in $\mathcal{EL}^-\mathcal{R}^+$ and contains 49523 axioms, 20528 concept names, and 20 role names.

2.6 InfoGlue

The InfoGlue OWL DL ontology is the by-product of a DL-based approach to support the comprehension and maintenance of software systems [RZH+07]. Program comprehension is seen as a knowledge intensive activity, requiring a large amount of effort to synthesize information obtained from different sources. The InfoGlue approach aims to reduce the comprehension effort by automatically identifying concept instances and their relations in different software artifacts. The ontology has $\mathcal{ALCH}$ expressivity and it contains both a TBox and an ABox. More information on InfoGlue can be found in Table 2.

2.7 Lehigh University Benchmark

The Lehigh University Benchmark (LUBM, [GHP03, GPH04, GPH05]) [7] ontology represents the structural organization of an university (with all departments).

[4] `http://www.cs.concordia.ca/FungalWeb/`
[5] `http://www.co-ode.org/galen/`
[6] `http://www.geneontology.org/`
[7] `http://swat.cse.lehigh.edu/projects/lubm/index.htm`

In this report LUBM will used with two different TBoxes (lite and normal) and 6 different ABox sizes ranging from 5 to 50 universities (with all departments). The original LUBM TBox is in $\mathcal{ELH}$. However, due to some preprocessing strategies (GCI absorption) some of the reasoners that are used in this report (e.g., RACER) disjunctions might be added to the axioms resulting in a TBox which is in $\mathcal{ALCH}$. In the case of LUBM another absorption is possible that avoids the addition of disjunctions but this is currently not supported by some reasoners (e.g., RACER). However, a slight modification to the original LUBM TBox can avoid the unnecessary addition of disjunctions. Thus, we decided to investigate two variants of LUBM, the original one, called LUBM, and the modified one, called LUBM-Lite. The characteristics of the LUBM KBs are described in detail in Table 2.

In the spirit of LUBM, another set of ABoxes representing the engineering department of the University of Rome was derived (including a LUBM-like TBox). Due to confidentiality problems, the benchmark ABoxes are not publicly accessible. Thus, they are not tested with all reasoners.

2.8 OntoCAPE

The OntoCAPE is an ontology developed based on CLiP, a comprehensive data model for process engineering. The ontology is organized to cover both common CAPE (Computer-Aided Process Engineering) concepts as well as those which are application/purpose-specific. Concretely, the former part presents concepts of chemical process materials and chemical process systems, as well as mathematical and computer software concepts which are often applied in various CAPE tasks. In the latter part, concepts that support applications including process design and modeling are considered. The OntoCAPE ontology is designed in the layered fashion, see [MYM07]. It contains 575 (in most cases primitive) concept definitions.

Originally, this TBox uses the DL $\mathcal{SHIQ}(\mathcal{D})$, i.e., concept constructors from $\mathcal{ALCQ}$ in combination with data types, role declarations for inverse roles and transitivity and domain and range restrictions for roles and attributes. Moreover, it contains GCIs and cyclic definitions. For testing some of the systems we had to use variants of the OntoCAPE TBox that use only concept constructors the respective inferences and services can handle. Different versions of the knowledge base exist (see the details in the benchmarking sections).

2.9 Pizza

An example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University. [8] The ontology is represented in $\mathcal{ALCH}f$ and it contains 173 axioms, 42 concept names, and 26 role names.

[8] `http://www.co-ode.org/resources/tutorials/`

2.10 Role

The Role or Bio-Zen[9] ontology is an ontology for the life sciences. In its current version, it is focused on the representation and mathematical modeling of molecular structures, biochemical and physiological processes and interaction networks. Bio-Zen is based on foundational ontologies and metadata standards (DOLCE, SKOS, Dublin Core). It is unique in that it unifies a high degree of ontological consistency with a maximum of flexibility and simplicity in its design, and it uses synonyms for roles in the ontology. The ontology is written in $\mathcal{SHIN}$ and contains 170 axioms, 47 concept names, and 85 role names.

2.11 SoftEng

The SoftEng ontology was created by a reverse engineering approach [ZRH06] where Java code is represented in an abstract way as a KB and DL inference services are used to reason about security concerns. It is represented in $\mathcal{L}^{-}\mathcal{H}R^{+}$ and it contains both a TBox and an ABox. More information on SoftEng can be found in Table 2.

2.12 University Ontology Benchmark

The UOBM or UOB ontology [MYQ+06] was derived from LUBM but has a more complicated TBox and ABox structure. It represents the structural organization of an university (with all departments). For benchmarking the variant based on OWL-Lite is used with 5 different ABox sizes ranging from 1 to 5 universities (with all departments). The characteristics of the UOBM KBs are shown in Figure 14.

2.13 Web Mining Ontologies

The two Web Mining ontologies (WebMin 1 and 2) are proprietary and were contributed by users of RACER. WebMin 1 and WebMin 2 are written in $\mathcal{ALEHf}(\mathcal{D}^{-})$. To a large extent, they use datatype values (e.g., strings). The characteristics of the Web Mining ontologies are shown in Table 2.

2.14 Wordnet

The Wordnet knowledge base (version 1.7.1)[10] is an OWL-DL KB representing the WordNet 1.7.1 lexical database. The characteristics of the Wordnet KB are given in Figure 7. Wordnet contains 84K concept names, 269K individual names, 548K individual assertions, and 304K role assertions.

[9] `http://neuroscientific.net/semantic`

[10] `http://taurus.unine.ch/files/wordnet171.owl.gz`

3 Standard Reasoning in Expressive DLs

With "standard reasoning", we refer to testing satisfiability and subsumption of concept descriptions, which are the most traditional reasoning services offered by description logic reasoning systems. In this section, we test two systems that have been developed within TONES and support standard reasoning: FaCT++ and Racer. We evaluate their performance against each other, and against the non-TONES reasoners PELLET and KAON2.

3.1 FaCT++

FaCT++ is a sound and complete DL reasoner designed as a platform for experimenting with new tableaux algorithms and optimisation techniques.[11] It incorporates most of the standard optimisation techniques, but also employs many novel ones.

DL systems take as input a knowledge base (equivalently an ontology) consisting of a set of axioms describing constraints on the conceptual schema (often called the TBox) and a set of axioms describing some particular situation (often called the ABox). They are then able to answer both "intensional" queries (e.g., regarding concept satisfiability and subsumption) and "extensional" queries (e.g., retrieving the instances of a given concept) w.r.t. the input knowledge base (KB). For the expressive DLs implemented in modern systems, these reasoning tasks can all be reduced to checking KB satisfiability.

When reasoning with a KB, FaCT++ proceeds as follows. A first *preprocessing* stage is applied to the KB when it is loaded into reasoner; it is normalised and transformed into an internal representation. During this process several optimisations (that can be viewed as a syntactic re-writings) are applied.

The reasoner then performs *classification*, i.e., computes and caches the subsumption partial ordering (taxonomy) of named concepts. Several optimisations are applied here, mainly involving choosing the order in which concepts are processed so as to reduce the number of subsumption tests performed.

The classifier uses a KB *satisfiability* checker in order to decide subsumption problems for given pairs of concepts. This is the core component of the system, and the most highly optimised one.

FaCT++ can be downloaded at the following address: `http://owl.man.ac.uk/factplusplus/`. Within TONES FaCT++ has been extended with new optimization techniques and to support $\mathcal{SROIQ}$, the logic underlying OWL 1.1.

3.2 Racer

Racer [HM01a, EHK+07, EKM+07] is under continuous development since 1998 (at the time of this writing, commercial support is available for two years). The system is used for ontology design and maintenance (offline usage of ontologies) as well as for using ontologies in running applications that rely on reasoning (online usage of ontologies). Since ontologies get larger and larger, and new application fields use ontologies these

[11] FaCT++ is available at `http://owl.man.ac.uk/factplusplus`.

days, the demands on system architecture ever increase. RACER has been designed for optimized TBox as well as optimized ABox reasoning.

Basically, the system implements the description logic $\mathcal{SHIQ}(\mathcal{D}n)$ with TBoxes and ABoxes (see [BBC+07] for details about syntax and semantics of description logics). All standard DL inference services for ontology design and maintenance are provided by RACER. In order to assist the creation of practical applications, the RACER system includes several extensions the development of which has been partially supported by TONES project. The wide spectrum of supported inference services, e.g., classification, answering grounded conjunctive queries (see Section 5.1), some non-standard inferences, abductive query answering (see Section 7) makes RACER unique.

Several interfaces are available for RACER. As usual, the reasoner supports file-based interaction as well as socket-based communication with end-user applications or graphical interfaces for ontology development and maintenance. Input can be specified in various syntaxes, e.g., KRSS (TCP), DIG 1.1 (HTTP), or OWL DL (HTTP). A parser for DIG 2.0 [TBK+06] is in preparation. As an extension to DIG 1.1, RACER already supports an XML-based interface for conjunctive queries. The specification of this interface is also proposed as part of DIG 2.0 with some slight modifications [TBK+06]. The RACER implementation of DIG 2.0 will support also expressive constraints. Unparsers from the internal meta model to a textual representation of ontologies are available for all syntaxes.

In particular, for DIG 2.0 it will be the case that not all syntactic constructs might be implemented by a certain reasoner. For instance, DIG 2.0 includes nominals as part of the TBox (this also holds for OWL DL). Currently, RACER fully supports nominals as part of ABoxes. Nominals in the TBox are approximated by concept names. For fully supporting the OWL 1.1. fragment of DIG 2.0, also acyclic role axioms have to be provided by the RACER implementation. It is well known, however, that for some purposes, even DIG 2.0 is not expressive enough. Further extensions are described in subsequent paragraphs.

Rule specifications are well-known (e.g., from the W3C SWRL specification[12]), but different systems support different semantics (for details of the RACER semantics for rules, see the RACER reference manual[13]). In RACER, rules can be seen as a convenient specification about how to extend the set of assertions in an ABox. In addition, rules can be used as named queries that can be reused in other queries. Rule design is also part of ontology design. Rule bodies can be checked for subsumption (grounded semantics). Rules in RACER can be specified with a KRSS or SWRL syntax.

In some sense, OWL is rather inexpressive in that it does not support constraints between attribute values of different individuals. For instance, in OWL it is not possible to state that Mike's brother, called John, is ten years older than Mike, and Mike is a car driver (and the ontology says that car drivers must be older than 18). Does this mean that, concerning the age, John is allowed to drive a car as well? RACER supports inequations about linear polynomials over the reals and over positive integers. In addition, RACER allows for expressing min/max restrictions over integers as well as (in)equalities over strings. If individuals are part of the ontology (and OWL even supports nominals in the TBox), consistency checking is an important issue at ontology-development time. At the time of this writing, constraints between different individuals are still not supported

[12] `http://www.w3.org/Submission/SWRL/`

[13] `http://www.racer-systems.com/products/racerpro/manual.phtml`

by the latest proposal for the new OWL language: OWL 1.1[14]. They are supported by DIG 2.0, however.

3.3 External Reasoners

3.3.1 PELLET

PELLET is an open source, OWL DL reasoner in Java, originally developed at the University of Marylands Mindswap Lab. PELLET is now commercially supported by Clark & Parsia LLC. PELLET is publicly available at http://pellet.owldl.com/.

Based on the tableaux algorithms developed for expressive Description Logics (DL), PELLET supports the full expressivity of OWL-DL, including reasoning about nominals (enumerated classes). As of version 1.4, PELLET supports all the features proposed in OWL 1.1, with the exception of n-ary datatypes. It also incorporates various optimization techniques described in the DL literature and contains several novel optimizations for nominals, conjunctive query answering, and incremental reasoning.

PELLET provides standard and cutting-edge reasoning services. In particular, PELLET provides all the standard reasoning services for ontologies, such as ontology consistency, concept satisfiability, concept subsumption, and instance checking. In addition, it also incorporates other more innovative services, such as Conjunctive ABox query answering, datatype reasoning, axiom pinpointing and debugging, among others.

3.3.2 KAON2

KAON2 is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies. It was produced by the joint effort of the following institutions: the Information Process Engineering (IPE) at the Research Center for Information Technologies (FZI), the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe, and the Information Management Group (IMG) at the University of Manchester.

The API of KAON2 is capable of manipulating OWL-DL ontologies. For reasoning, KAON2 supports the SHIQ(D) subset of OWL-DL. This includes all features of OWL-DL apart from nominals (also known as enumerated classes). Since nominals are not a part of OWL Lite, KAON2 supports all of OWL Lite.

KAON2 also supports the so-called DL-safe subset [3] of the Semantic Web Rule Language (SWRL). The restriction to the DL-subset has been chosen to make reasoning decidable.

KAON2 supports answering conjunctive queries, albeit without true non-distinguished variables. This means that all variables in a query are bound to individuals explicitly occurring in the knowledge base, even if they are not returned as part of the query answer.

Contrary to most currently available DL reasoners, such as FaCT, FaCT++, RACER, DLP or PELLET, KAON2 does not implement the tableaux calculus. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base

[14] http://www.webont.org/owl/1.1/

Ontology	Expressivity	Size in KByte (OWL file)
Galen	$\mathcal{SH}f$	2300
GODaily	$\mathcal{EL}^-\mathcal{R}^+$	9800
Indivi	$\mathcal{LU}$	1
Neuron	$\mathcal{L}^-$	538
Pizza	$\mathcal{ALCH}f$	20
Role	$\mathcal{SHIN}$	11

Figure 1: Ontologies used for TBox classification tests.

to a disjunctive datalog program. The system is available for download at the following URL: http://kaon2.semanticweb.org/.

3.4 Test Setup

We have tested TBox classification with respect to the collection of OWL ontologies shown in Figure 1. We have selected ontologies from quite different sources such that a reasoning system cannot benefit from particularities of a single test ontology. Furthermore, we tried to ensure representativity of our collection by:

- Having a broad range of expressivity: from $\mathcal{L}^-$ to $\mathcal{SHIN}$.
- Having ontologies of different input size: from 1 KByte to almost 10 MByte.

While, at a first glance, 10 MByte does not seem a lot nowadays, it turned out, that (expressive) ontologies of that size can already take current reasoning systems to their limits - with respect to classification.

To conduct tests with FaCT++ and Racer, we have transformed each OWL ontology into a reasoner specific Lisp-like format respectively. For the sake of completeness we will describe the transformation process for both reasonings.

For FaCT++ the transformation is mandatory, since it does not support OWL directly. A conversion tool is provided as a Web service at `http://www.mygrid.org.uk/OWL/Converter`. Unfortunately, this service was unavailable several times while performing our tests.

For Racer we performed a conversion of each ontology file as follows:

```
(owl-read-file "KB.owl" )
(save-kb "KB.racertbox")
```

Conversion times for all ontologies are shown in Figure 2. Notice, that this table is not intended as a comparison between Racer and FaCT++. In fact, it would not be a fair competition anyways, since for FaCT++ the whole ontology was sent via the Internet to a web service and back, while for Racer everything was done locally.

All the tests were performed on an Intel Pentium 4 with 2.80 GHz and 1 GB main memory with a Linux-based operating system (Linux kernel for version 2.6.20 on x86 32bit). We have used JRE 1.5 to run the Java-based reasoning systems.

To enable reproducibility of our tests, we will show next, how we made each reasoning system classify an input ontology.

Ontology	Convert to Fact++ (ms)	Convert to Racer (ms)
GODaily	39138	31980
Galen	6946	6360
Indivi	109	180
Neuron	5703	2880
Pizza	164	420
Role	1560	1320

Figure 2: Conversion times for FaCT++ and Racer.

```
(racer-read-file "Ontology.racertbox" :KB-NAME MyOnto)
(classify-tbox )
```

Figure 3: Input file for TBox classification with Racer.

- Racer : For each ontology we run the command *./RacerPro -f infile.racer* on the input file shown in Figure 3.
- FaCT++ : For each ontology, we have modified the standard *options.default* file by adding 'TBox=Ontology.tbox' and then run the command *./FaCT++ options.default*
- PELLET : For each ontology we run the command *java -Xss4m -Xms30m -Xmx200m -jar lib/pellet.jar "Ontology.owl"*
- KAON2 : Since we have found no way to perform TBox classification with the standard KAON2 package, we used the small Java program shown in Figure 4. Our program uses the KAON2 API to load an ontology file and compute the subsumption hierarchy.

For our tests, we did not only measure the time necessary to compute the classification of each ontology, but also (an indication) for the memory footprint of each reasoner. This is done by computing the amount of free main memory plus the free swap space, like 20 times a second, while the reasoner is processing the input. Since no other process was running on the machine during the tests, we claim that we can read off an indication of the maximum memory usage of each reasoning system for every ontology.

3.5 Test Results

This sections presents the results for standard TBox reasoning. First of all, we want to emphasize that it is hard to interpret these results in detail, when we have all reasoning systems only as a black box.

Figure 5 shows the time needed for classification of each ontology. For all ontologies but Neuron, PELLET is the slowest of all compared reasoners. For Neuron, PELLET throws a NullPointer-Exception, that might be related to one import file in the ontology. The results of FaCT++ and Racer are usually close to each other. Just for Galen and GODaily, the differences are more clear for each reasoner respectively.

We also compared the actually classification hierarchy for each ontology and it turned out, that there are some differences in the Pizza ontology, that are mainly caused by

```
[...]
KAON2Connection connection=KAON2Manager.newConnection();
DefaultOntologyResolver resolver=new DefaultOntologyResolver();

resolver.registerReplacement(ontologyURL,KAON2Directory+"KBs/myOntology.owl");
Ontology ontology=connection.openOntology(ontologyURL, new HashMap<String,Object>());
Object hierarchy=ontology.createReasoner().getSubsumptionHierarchy();
[...]
```

Figure 4: Java file for TBox classification with KAON2.

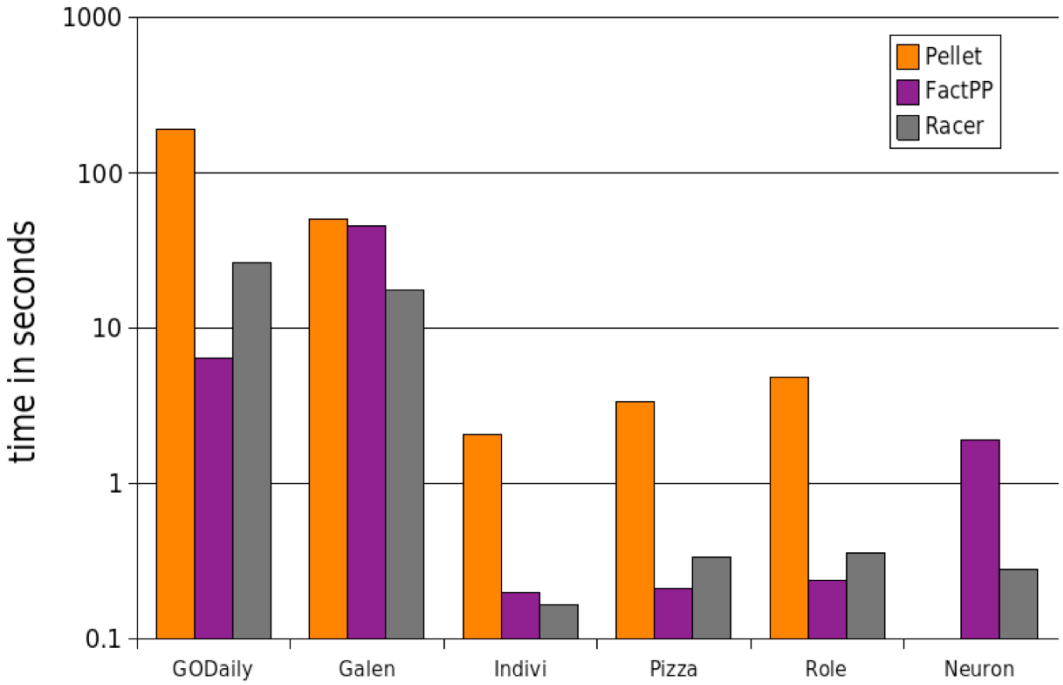


Figure 5: TBox classification - time (in sec).

different handling of nominals. Furthermore, FaCT++ has a problem with the Role ontology by missing one concept parent.

Figure 6 shows (an indication of) the memory needed for classification of each ontology. For all ontologies, PELLET needs the highest amount of memory to compute the classification. This might well be related to the fact that it is written in Java.

With the exception of GODaily, Racer has the lowest memory footprint of all tested reasoning systems. FaCT++ needs an average amount of memory, while the results are usually more close to the one of Racer, than of PELLET.

Originally we intended to provide test results for KAON2 as well. After running all the tests, we decided to not mention KAON2 in the result charts. The reason is, that for the majority of ontologies, KAON2 has thrown some unexpected exceptions or complained about the syntax of the OWL input. Since all other reasoners were able to work with the input files, we conjecture, that the fault is with KAON2. For the sake of completeness, we list the errors that KAON2 produced for each ontology here:

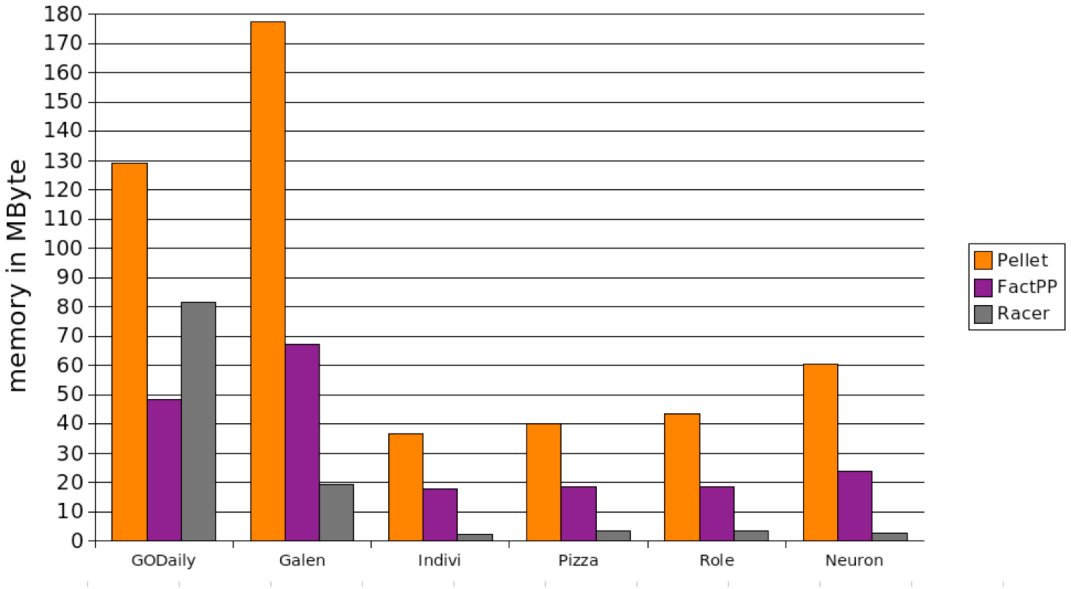


Figure 6: TBox classification - maximum memory usage (in MByte).

- **Indivi**: KAON2Exception - Nominals are not supported yet
- **Pizza**: KAON2Exception - ObjectHasValue is not supported yet
- **Neuron**: KAON2Exception - Cannot parse the descriptor

4 Standard Reasoning in Lightweight DLs

4.1 CEL

CEL is the first reasoner for the lightweight description logic $\mathcal{EL}^+$, supporting as its main reasoning task the computation of the subsumption hierarchy induced by $\mathcal{EL}^+$ ontologies. The most distinguishing feature of CEL is that, unlike other modern DL reasoners, it implements a polynomial-time algorithm. The supported Description Logic $\mathcal{EL}^+$ offers a selected set of expressive means that are tailored towards the formulation of medical and biological ontologies.

4.2 Test Setup

The aim of our experiments in this section is to demonstrate scalability of the polytime algorithm and reasoner in comparison to highly complex algorithms and reasoners for much more expressive description logics. To this end, we measure the time required for building up the concept hierarchy, i.e., classification time. We have performed a number of experiments with the lightweight reasoner CEL (version 1.0b), as well as the OWL-DL reasoners FaCT++ (version 1.1.0) and Racer (version 1.9.1b). All the experiments have been carried out on a standard PC: 2.8 GHz Pentium-4 processor and 1 GB of physical memory.

	$\mathcal{O}^{\text{NCI}}$	$\mathcal{O}^{\text{NotGalen}}$	$\mathcal{O}^{\text{Galen}}$	$\mathcal{O}^{\text{Snomed}}$
♯Concept names	27 652	2,748	23 136	379 691
♯Role names	70	413	950	62
♯Concept axioms	46 800	3 937	35 531	379 691
♯Role axioms	140	442	1 016	11 + 1
CEL	6.74	8.25	527.30	1 671.23
FaCT++	3.11	136.63	*unattainable*	3 206.69
Racer	24.74	13.43	*unattainable*	3 529.43

Table 1: Lightweight Ontologies and Classification Times

We consider some large-scale medical ontologies that are represented in $\mathcal{EL}^+$. SNOMED CT and NCI have been entirely designed in this lightweight logic, while Galen is represented in $\mathcal{SHIF}$. Large part (97.75%) of the full Galen is nevertheless expressible in the lightweight description logic $\mathcal{EL}^+$, which we used in our experiments. For comparison purposes, we also considered the $\mathcal{EL}^+$ fragment of NotGalen (a.k.a. simplified Galen). For detailed descriptions of these ontologies, please refer to Deliverable D14. To make the experiments self-contained, however, we also include some concise information on the structure and size of these ontologies in the upper part of Table 1. Note in the case of $\mathcal{O}^{\text{Snomed}}$ that the only existing right-identity rule was not passed to Racer, as it does not support this.

In order to suppress noise in our experimental results, we performed each experiment for five times, i.e., five runs of each reasoner on each ontology. The measured classification times were sorted, and the average of the three middle values was calculated.

4.3 Test Results

These average classification times are shown in second in the lower part of Table 1, where all classification times are in seconds and *unattainable* means the reasoner either failed due to memory exhaustion or did not terminate after 24 hours. Interesting remarks on our experimental results are highlighted in order:

- CEL outperforms all the reasoners in all benchmarks, except for the case of $\mathcal{O}^{\text{NCI}}$, where CEL is slower than FaCT++ but still faster than Racer. The main reason for this is that $\mathcal{O}^{\text{NCI}}$ comprises *only* primitive concept definitions, for which the optimization technique of "completely defined concepts" is effective. This technique has been implemented in FaCT++ but not in Racer and CEL.
- The largest ontology $\mathcal{O}^{\text{Snomed}}$ is classified by CEL in only half the time needed by FaCT++ and Racer.
- CEL is the only reasoner that can classify $\mathcal{O}^{\text{Galen}}$.

The main reason that no reasoners based on a tableau algorithm can classify $\mathcal{O}^{\text{Galen}}$ is that the ontology is highly complex with thousands of GCIs, many of which cannot

be absorbed into concept definitions or role constraints. CEL however does not show degradation of performace on this kind of ontology, and we view this as an indication that CEL's computational behavior is more robust than that of tableau-based reasoners, of course, given that the ontology is formulated in the lightweight language $\mathcal{EL}^+$. Finally, we could view the CEL reasoner as an empirical evidence of scalibility of the polytime algorithm described in Deliverable D13.

5 Query Answering

5.1 RACER

5.1.1 Tool Description

RACER has already been described in the previous section. In addition to the basic retrieval inference service, expressive query languages are required in practical applications. Well-established is the class of conjunctive queries. Reasoning algorithms, optimization techniques and systems for answering conjunctive queries have been extensively described in [CGG+07a]. We give a short introduction to conjunctive queries here for the sake of completeness, though.

A *conjunctive query* consists of a *head* and a *body*. The head lists variables for which the user would like to compute bindings. The body consists of query atoms (see below) in which all variables from the head must be mentioned. If the body contains additional variables, they are seen as existentially quantified. A query answer is a set of tuples representing bindings for variables mentioned in the head. A query is a structure of the form $ans(X_1, \ldots, X_n) \leftarrow atom_1, \ldots, atom_m$.

Query atoms can be *concept* query atoms ($C(X)$), *role* query atoms ($R(X, Y)$), *same-as* query atoms ($X = Y$) as well as so-called *concrete domain* query atoms. The latter are introduced to provide support for querying the concrete domain part of a knowledge base and will not be covered in detail here. Complex queries are built from query atoms using boolean constructs for conjunction (indicated with comma) or union ($\vee$).

In *standard* conjunctive queries, variables (in the head and in query atoms in the body) are bound to (possibly anonymous) domain objects (see the Section 5.2). In so-called *grounded* conjunctive queries, $C(X)$, $R(X, Y)$ or $X = Y$ are true if, given some bindings α for mapping from variables to *individuals mentioned in the Abox* $\mathcal{A}$, it holds that $(\mathcal{T}, \mathcal{A}) \models \alpha(X) : C$, $(\mathcal{T}, \mathcal{A}) \models (\alpha(X), \alpha(Y)) : R$, or $(\mathcal{T}, \mathcal{A}) \models \alpha(X) = \alpha(Y)$, respectively. In grounded conjunctive queries the standard semantics can be obtained for so-called tree-shaped queries by using corresponding existential restrictions in query atoms. Due to space restrictions, we cannot discuss the details here. RACER supports grounded conjunctive queries [WM05]. The language implemented in RACER is called nRQL (pronounce: "niracle" and hear it as "miracle").

5.1.2 Test Setup

We selected two sets of knowledge bases for benchmarking. The first set consists of ten KBs with large ABoxes that were derived from various applications of DL technology

Knowledge Base	TBox Logic	Concept Names	Roles	Axioms
SoftEng	$\mathcal{L}^-\mathcal{H}$	37	30	76
SEMINTEC	$\mathcal{FL}_0 f$	59	24	345
FungalWeb	$\mathcal{ALCH}(\mathcal{D})$	3 601	77	7 209
InfoGlue	$\mathcal{ALCH}$	41	37	83
WebMin 1	$\mathcal{ALEH}f(\mathcal{D}^-)$	444	175	1 714
WebMin 2	$\mathcal{ALEH}f(\mathcal{D}^-)$	520	204	1 893
LUBM	$\mathcal{ALCH}$	43	41	85
FERMI	$\mathcal{L}$	5 136	15	10 265
UOBM-Lite	$\mathcal{ALCH}f$	51	49	101
VICODI	$\mathcal{L}^-\mathcal{H}$	194	10	387

Knowledge Base	Abox Logic	Inds	Ind. Assertions	Role Assertions
SoftEng	$\mathcal{L}^-\mathcal{H}R^+$	6 735	5 595	25 896
SEMINTEC	$\mathcal{FL}_0 f$	17 941	17 941	41 174
FungalWeb	$\mathcal{ALCH}(\mathcal{D})$	12 556	12 705	1159
InfoGlue	$\mathcal{SH}$	15 464	15 937	88 316
WebMin 1	$\mathcal{ALCH}f(\mathcal{D}^-)$	1 427	9 193	1 146
WebMin 2	$\mathcal{ALCH}f(\mathcal{D}^-)$	6 532	28 676	15 915
LUBM	$\mathcal{SH}(\mathcal{D}^-)$	17 174	51 207	49 336
FERMI	$\mathcal{EL}$	700	9 998	650
UOBM-Lite	$\mathcal{SH}f(\mathcal{D}^-)$	5 674	10 790	11 970
VICODI	$\mathcal{L}^-\mathcal{H}$	16 942	16 942	36 704

Table 2: Characteristics of the 10 application KBs used for Abox benchmarking.

within the semantic web community. The second set contains three knowledge bases with very large ABoxes. The ontologies are briefly described in Section 2.

The first set of OWL knowledge bases contains relatively simple TBoxes but large ABoxes. The characteristics of these KBs are summarized in Table 2. The second column in Table 2 characterizes the TBox logic determined by RACER after TBox classification. The TBox logic of the input file might be different because RACER might add disjunctions to the processed KB due to GCI absorption (e.g., this is the case for the LUBM KB). The third column shows the number of concept names, the fourth the number of roles, and the fifth the number of TBox axioms. The sixth column gives the ABox logic determined by RACER after testing ABox consistency. The seventh to ninth columns show the number of individuals, individual assertions, and role assertions. It is interesting to note that the ABox logic is sometimes slightly more complex than the TBox logic.

The TBox/ABox logic is indicated using the standard DL terminology (see [Baa03] for details). We additionally denote the DL supporting only conjunction and primitive negation by $\mathcal{L}$, whereas $\mathcal{L}^-$ stands for $\mathcal{L}$ without primitive negation. Both variants of $\mathcal{L}$ also admit simple concept inclusions whose left-hand sides consist only of a name. The notation "$(\mathcal{D})$" is used to denote the use of concrete domain expressiveness (see [HM01b, Baa03] for details). The occurrence of "$(\mathcal{D})$" is caused by OWL datatype properties that are restricted by number restrictions (and RACER applies concrete domain reasoning

to these constructs) whereas "($\mathcal{D}^-$)" denotes "($\mathcal{D}$)" without number restricted datatype properties. Furthermore, the use of functional roles is denoted as f and of transitive roles as R^+ (or $\mathcal{S}$).

It is also important to mention that RACER computes the TBox/ABox logic of a KB by analyzing it in detail. For instance, if a KB declares a transitive (inverse) role (as in the case of LUBM) but never uses this role within an axiom or, in the case of an inverse role, there does not exist an interaction between a role and its inverse (e.g., $\exists R.(\forall R^-.C)$), then the TBox logic does not refer to transitive or inverse roles. The same methodology is applied to ABoxes but the logic of the ABox is always at least as expressive as the one of its TBox.

For implementing sound and complete inference algorithms, tableau-based algorithms are known to provide a powerful basis. Nowadays, almost all practical systems for (supersets of) $\mathcal{SHIQ}(\mathcal{D}_n)^-$ employ highly optimized versions of tableau-based algorithms. It should be emphasized that the research approach behind RACER is oriented towards applications and strives to provide a good overall performance and reliability. The term "applications" refers in this context to knowledge bases which were generated or contributed by persons or organizations using DL technology. In this context we deliberately ignore whether a knowledge base is considered as "synthetic" or "realistic" because the presented optimization techniques should work well for any kind of knowledge base.

We evaluate the optimization techniques presented in the following sections in the context of ABox realization and instance retrieval problems for application knowledge bases. In particular, we consider applications for which ABox reasoning is actually required, i.e., implicit information must be derived from ABox statements and TBox axioms, and ABoxes are not only used to store relational data. Thus, instance retrieval cannot be reduced to computing queries for (external) relational databases (see, e.g., [BB93], [Bre95], [BHT05]).

For evaluation purposes we do not extensively compare query answering speed with other DL systems but investigate the effect of optimization techniques that could be exploited by any (tableau-based) DL inference system that already exists or might be built. In addition, from a methodological point of view, performance comparisons with other systems (e.g., see [MS06]) are not as informative as one might think. The reason is that, in general, it is hard to operate systems in the same mode with optimization techniques switched on and off. In addition, whether a certain system seems to be slow for some specific knowledge base and query might be the result of various effects that can hardly be tracked down from an external point of view, and those effects tell us nothing about the usefulness of the optimization techniques under investigation. For the sake of completeness we compare the standard settings of the RACER engine with the standard settings of the KAON2 system and rerun the benchmarks used in [MS06] with a newer version of RACER (see below).

Recently, various optimization techniques for partitioning ABoxes into independent parts and/or creating condensed (summary) ABoxes [FKM$^+$06, GH06, DFK$^+$07] have been reported. The advantages of ABox partitioning are not the topic of this investigation. RACER employs a straight-forward ABox partitioning technique [HM99] which is based on pure connectedness of graphs because an ABox can be viewed as a possibly cyclic graph defined by a set of role assertions. More precisely, if an ABox can be partitioned into

independent parts which are also not related via assertions involving concrete domains), RACER employs a divide-and-conquer strategy which applies the algorithms described below to each partition and combines the results afterwards.

We like to emphasize that the optimization techniques mentioned in this section are still very useful in the presence of more refined ABox partitions schemes because they are still applicable to single partitions. Moreover, our techniques are even more vital for scenarios where ABoxes cannot be partitioned or still contain large partitions. All these techniques are applicable in two general application scenarios: (i) ABox realization and (ii) instance retrieval or, more general, query answering (without precomputing a realization). The techniques are evaluated on the basis of these two scenarios.

For the evaluation we selected a set of 10 application knowledge bases (see also Section 5.1.2 for more details) with usually small and simple TBoxes but large ABoxes whose sizes are varying between 700 and 18K individuals, 9K and 51K individual assertions, and between 650 and 88K role assertions. Furthermore we tested three ontologies with very large ABoxes. LUBM [GPH05] is used with two different TBoxes (LUBM-Lite and LUBM) and 6 different ABox sizes (5 to 50 universities) which result for 50 universities in 1 082K individuals, 3 355K individual assertions, and 3 298K role assertions. UOBM [MYQ+06] was derived from LUBM. It exhibits a more complicated TBox and ABox structure but smaller ABox sizes (1-5 universities) which results for 5 universities in 138K individuals, 509K individual assertions, and 563K role assertions. Wordnet (version 1.7.1) is an OWL-DL KB representing the WordNet 1.7.1 lexical database and contains 84K concept names, 269K individual names, 548K individual assertions, and 304K role assertions. All these ontologies are tested with various sets of optimization settings which disable or enable particular optimization techniques.

LUBM queries are modeled as grounded conjunctive queries referencing concept, role, and individual names from the TBox. Below, LUBM queries 9 and 12 are shown in order to demonstrate LUBM query answering problems – note that 'www.University0.edu' is an individual and *subOrganizationOf* is a transitive role. Please refer to [GHP03, GPH04, GPH05] for more information about the LUBM queries.

$$\begin{aligned} Q9 : ans(x, y, z) \leftarrow\ & Student(x), Faculty(y), Course(z), \\ & advisor(x, y), takesCourse(x, z), teacherOf(y, z) \\ Q12 : ans(x, y) \leftarrow\ & Chair(x), Department(y), memberOf(x, y), \\ & subOrganizationOf(y, \text{www.University0.edu}) \end{aligned}$$

In order to investigate the data description scalability problem, we used a TBox provided with the LUBM benchmarks. The TBox declares (and sometimes uses) inverse and transitive roles as well as domain and range restrictions, but no number restrictions, value restrictions or disjunctions. Among other axioms, the LUBM TBox contains axioms that express necessary and sufficient conditions for some concept names. For instance, the TBox contains an axiom for *Chair*: $Chair \equiv Person \sqcap \exists headOf.Department$. For evaluating our optimization techniques for query answering we consider runtimes for a whole query set (queries 1 to 14 in the LUBM case).

If grounded conjunctive queries are answered in a naive way by evaluating subqueries in the sequence of syntactic notation, acceptable answering times can hardly be achieved.

For efficiently answering queries, a query execution plan is determined by a cost-based optimization component (c.f., [GMUW02, p. 787ff.], see also [CM77]) which orders query atoms such that queries can be answered effectively. Query execution plans are specified in the same notation as queries (whether a query is seen as an execution plan will be clear from context). We assume that the execution order of atoms is determined by the order in which they are textually specified.

Let us consider the execution plan $ans(x, y) \leftarrow C(x), R(x, y), D(y)$. Processing the atoms from left to right will start with the atom $C(x)$. Since there are no bindings known for the variable x, the atom $C(x)$ is mapped into a query $instance\_retrieval(C, \mathcal{A}, individuals(\mathcal{A}))$. The elements in the result set of the retrieval query are possible bindings for x. $C(x)$ is called a *generator*. The next query atom in the execution plan is $R(x, y)$. There are bindings known for x but no bindings for y. Thus, $R(x, y)$ is also a generator (for y-bindings). Given the atom $R(x, y)$ is handled by a role filler query for each binding of x, there are possible bindings for y generated. Afterwards, the atom $D(y)$ is treated. Since there are bindings for y available, the atom is mapped to an instance test (for each binding). We say, the atom $D(y)$ acts as a *tester*.

Determining all bindings for a variable (with a generator) is much more costly than verifying a particular binding (with a tester). Treating the one-place predicates *Student*, *Faculty*, and *Course* from query $Q9$ (see above) as generators for bindings for corresponding variables results in a combinatorial explosion (cross product computation). Optimization techniques are required that provide for efficient query answering in the average case.

As outlined earlier we evaluate the 10 application KBs using the inference service of Abox realization which heavily relies on the techniques introduced in the previous sections. This kind of ABox indexing w.r.t. to concept names is stress-testing these techniques and it is especially suitable if a sufficient number of specific benchmark queries for the selected ontologies are not available. Another argument for realization are RDF query languages such as SPARQL, which heavily relies on concept names for querying. In case a reasonable number of queries were available (as in the case of LUBM and UOBM) we additionally tested these KBs with these sets of queries (as detailed in Section 5.1.3).

Experimental Settings All experiments were conducted by switching on or off selected optimization techniques (as introduced in the previous sections) in order to assess the positive (and sometimes also negative) impact of these techniques on the runtimes. The tests were conducted on a Sun server V890 with 8 dual core processors and 64 GB of main memory (although all these tests usually require less than 2 GB of memory usage and each test was executed on a single processor). For each setting the average runtimes of the 10 application KBs were sequentially computed (without restarting RACER) where each KB test was repeated 5 times. The runtimes using the "standard" optimization setting of RACER are presented in Table 3. The second column shows the time for loading the KB, the third for classification, and the fourth for realization (including the time for the initial Abox consistency test). It can be clearly seen that the classification times can be neglected (as expected) and the realization times vary between less than 2 seconds and almost 3 minutes (since TBox classification times are neglectable, not all TBoxes mentioned here have been used in Section 3). The average runtimes are usually inflated

Knowledge Base	Load	TBox	Abox
SoftEng	7.43	0.03	5.61
SEMINTEC	7.25	0.07	18.6
FungalWeb	3.20	1.71	59.2
InfoGlue	28	0.04	161
WebMin 1	1.57	0.90	14.8
WebMin 2	8.63	0.74	104
LUBM	14.2	0.04	46.4
FERMI	7.56	3.35	1.33
UOBM-Lite	4.73	0.05	61.6
VICODI	5.65	0.07	34.1

(Load = load time, TBox = TBox classification time, Abox = Abox realization time)

Table 3: Processing times of application KBs using the std. opt. setting (in secs).

by the overhead caused by garbage collection.

For the evaluation of the application KBs the following 11 parameters were used to switch optimization techniques on or off. The parameters are described in detail in [CGG+07a] (see also [MHW06]).

P1 Individual pseudo model merging (see [CGG+07a, Section 2.1.2]): switched on by default.

P2 ABox contraction (see [CGG+07a, Section 2.2.1]): switched on by default.

P3 Sets-of-individuals-at-a-time instance retrieval (see [CGG+07a, Section 2.3]): switched on by default. If it is switched off, linear instance retrieval is selected.

P4 ABox completion (see [CGG+07a, Section 2.2.3]): switched on by default.

P5 ABox precompletion (see [CGG+07a, Section 2.2.4]): switched on by default.

P6 Binary instance retrieval (see [CGG+07a, Section 2.2.5]): switched on by default.

P7 Dependency-based instance retrieval (see [CGG+07a, Section 2.2.6]): switched on by default.

P8 Static index-based instance retrieval (see [CGG+07a, Section 2.3]): switched off by default (time and memory demands can be excessive in general).

P9 Dynamic index-based retrieval (see [CGG+07a, Section 2.3.1]): switched off by default (time and memory demands can be excessive in general).

P10 OWL-DL datatype properties simplification (see [CGG+07a, Section 2.3.2]): switched on by default.

P11 Re-use of role assertions for existential restrictions (see [CGG+07a, Section 2.3.3]): switched on by default.

On the basis of these 11 parameters we created 17 different benchmark settings as shown in Table 4. The rows in Table 4 define the following settings numbered 1-17 from top to bottom. These settings were selected to demonstrate the positive (and sometimes also negative) effect of optimization techniques. Many settings switch off only one optimization technique (compared to the standard setting) but several settings switch off up to three techniques which are interrelated to one another.

1. Standard setting with only static (P8) and dynamic index-based retrieval (P9) disabled. All the following settings are based on this standard setting.
2. ABox completion (P4) switched off.
3. ABox precompletion (P5) switched off.
4. ABox completion (P4) and precompletion (P5) switched off.
5. Individual pseudo model merging (P1) switched off.
6. ABox contraction (P2) switched off.
7. Sets-of-individuals-at-a-time (P3) switched off.
8. Sets-of-individuals-at-a-time (P3) and binary instance retrieval (P6) switched off.
9. Sets-of-individuals-at-a-time (P3), binary instance retrieval (P6), and dependency-based instance retrieval (P7) switched off.
10. OWL-DL datatype simplification (P10) switched off.
11. Re-use of role assertions (P11) switched off.
12. Static index-based instance retrieval (P8) switched on.
13. Dynamic index-based instance retrieval (P9) switched on.
14. Dependency-based instance retrieval (P7) switched off.
15. Binary instance retrieval (P6) switched off.
16. ABox precompletion (P5) and dependency-based instance retrieval (P7) switched off.
17. ABox precompletion (P5) and binary instance retrieval (P6) switched off.

5.1.3 Test Results

In this section we discuss the impact of the optimization techniques mentioned previously by considering runtimes for the traditional inference service of ABox realization and for KB specific queries. The runtimes we present here are used to demonstrate the order of magnitude of time resources that are required for solving inference problems when the complexity of the input problem is increased. They allow us to analyze the impact of the presented optimization techniques.

Setting	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
1	√	√	√	√	√	√	√	×	×	√	√
2	√	√	√	×	√	√	√	×	×	√	√
3	√	√	√	√	×	√	√	×	×	√	√
4	√	√	√	×	×	√	√	×	×	√	√
5	×	√	√	√	√	√	√	×	×	√	√
6	√	×	√	√	√	√	√	×	×	√	√
7	√	√	×	√	√	√	√	×	×	√	√
8	√	√	×	√	√	×	√	×	×	√	√
9	√	√	×	√	√	×	×	×	×	√	√
10	√	√	√	√	√	√	√	×	×	×	√
11	√	√	√	√	√	√	√	×	×	√	×
12	√	√	√	√	√	√	√	√	×	√	√
13	√	√	√	√	√	√	√	×	√	√	√
14	√	√	√	√	√	√	×	×	×	√	√
15	√	√	√	√	√	×	√	×	×	√	√
16	√	√	√	√	×	√	×	×	×	√	√
17	√	√	√	√	×	×	√	×	×	√	√

Table 4: Composition of the selected 17 different optimization settings. Setting 1 is the standard setting, √ = switched on, × = switched off.

Evaluation Using the Realization Inference Service The first series of evaluations were performed with most of the KBs introduced in Section 2. For all KBs the direct types for all individuals mentioned in the associated ABox were computed and verified. Each test was performed with all 17 settings described above. Each ABox realization was repeated five times and the average of these five runs is shown in the Tables 5 and 6. In the following we analyze the results by focusing on (i) the best and worst settings per KB, and (ii) for each setting the KB having the most positive and negative impact.

The SoftEng KB is only affected by S5 which disables individual pseudo model merging and results in an increase of runtime by a factor of 100. A similar observation holds for the SEMINTEC KB, which timed out after 1000 secs. Its best runtimes are for S7-S9, which switch the sets-of-individuals-at-a-time technique off. This indicates a 50% overhead for this technique.

The FungalWeb KB also timed out for setting S5. Otherwise it remains mostly unaffected if 10% variations are ignored.

The InfoGlue KB's best setting is S15, which switches off binary instance retrieval. This indicates that the partitioning scheme only causes overhead in this case. The second-best one is S2 (no completion). This indicates that the completion tests are mostly unsatisfiable and thus wasted due to the incompleteness of this technique. InfoGlue timed out for S3-S5, S7-S9, and S16-S17. The size of this KB and its TBox/ABox logics ($\mathcal{ALCH}/\mathcal{SH}$) explain the timeout for S3-S4, which switch off the precompletion technique and cause the overhead of recomputation of assertions making up the precompletion. A similar effect as for SEMINTEC also occurs for InfoGlue for S5. The disabled sets-of-individuals-at-a-time technique explains the timeout for S7-S9 because the then enabled

Knowledge Base	S1	S2	S3	S4	S5	S6	S7	S8
SoftEng	5.6	5.4	5.4	5.5	480	5.4	6.0	5.7
SEMINTEC	18.6	18.9	17.6	18.9	1000	18.8	12.2	11.5
FungalWeb	59.2	66.6	59.1	58.9	461	59.0	64.5	53.8
InfoGlue	160	113	1000	1000	1000	161	1000	1000
WebMin 1	14.8	14.9	14.5	15.0	45.4	14.8	13.2	13.5
WebMin 2	104	57.7	215	215	1000	57.8	124	67.9
LUBM	46.4	49.9	133	136	1000	58.8	185	189
FERMI	1.33	1.33	1.3	1.37	6.90	1.31	1.19	1.28
UOBM-lite	61.6	68.4	914	935	834	61.6	1000	1000
VICODI	34.1	22.3	23.5	23.0	1000	33.0	21.8	26.0

S1 = standard, S2 = completion off, S3 = precompletion off,
S4 = completion+precompletion off, S5 = ind. pseudo model merging off,
S6 = contraction off, S7 = sets-of-inds-at-a-time off,
S8 = sets-of-inds-at-a-time+binary instance retrieval off.

Table 5: Abox realization with opt. settings 1-8 (in secs, timeout after 1000 secs).

Knowledge Base	S9	S10	S11	S12	S13	S14	S15	S16	S17
SoftEng	5.67	5.42	5.45	5.47	5.45	5.6	5.5	6.45	6.23
SEMINTEC	11.6	18.3	16.2	18.9	18.8	18.1	18.0	18.9	22.9
FungalWeb	53.2	59.4	56.7	59.8	59.5	60.2	57.8	65.3	59.5
InfoGlue	1000	133	158	162	159	207	96.0	1000	1000
WebMin 1	13.5	22.3	14.8	14.8	14.8	15.2	14.4	14.9	14.5
WebMin 2	68.0	140	55.2	57.7	88.3	78.1	64.3	202	200
LUBM	191	68.7	66.6	59.3	58.7	59.1	40.6	170	200
FERMI	1.19	1.27	1.2	1.3	1.32	1.34	1.36	1.27	1.26
UOBM-lite	1000	66.8	66.2	61.5	61.8	55.9	58.1	1000	915
VICODI	25.7	33.1	21.0	33.2	32.9	23.7	28.6	29.8	23.4

S9 = sets-of-inds-at-a-time+binary+dependency-based instance retrieval off,
S10 = datatype simplification off, S11 = Re-use of role assertions off,
S12 = static index-based instance retrieval on,
S13 = dynamic index-based instance retrieval on,
S14 = dependency-based instance retrieval off, S15 = binary instance retrieval off,
S16 = precompletion+dependency-based instance retrieval off,
S17 = precompletion+binary instance retrieval off.

Table 6: Abox realization with opt. settings 9-17 (in secs, timeout after 1000 secs).

linear instance retrieval causes too much overhead.

The WebMin 1 KB remains mostly unaffected except by S5 (3 times slower) and S10 (50% slower), which disables the datatype simplification. The best settings for the WebMin 2 KB are S2, S6, S11, where the overhead of these techniques is saved, and S12, where the static index-based instance retrieval compensates for the overhead of the other techniques. WebMin 2's worst settings are S3-S4, which switch the precompletion off.

Due to its size LUBM timed out for S5 and its best runtime is for S15, which is 10% faster than the standard one. A factor of 4 in the increase of the runtime can be noticed for S7-S9 due to the overhead of linear instance retrieval.

The observations for the FERMI KB are similar to SoftEng, although S5 only causes an increase of a factor of 5. This is due to the very simple structure of this KB.

For UOMB-Lite the best is S14, which disabled dependency-based instance retrieval. S3-S4 cause an increase of a factor of 15 because they switch off the precompletion. A similar observation can be made for S5. Both can be explained by the size of the KB. S7-S9 timed out because sets-of-individuals-at-a-time was disabled and linear instance retrieval enabled.

The VICODI KB timed out for setting S5 and shows a variation of up to 50% in the other settings.

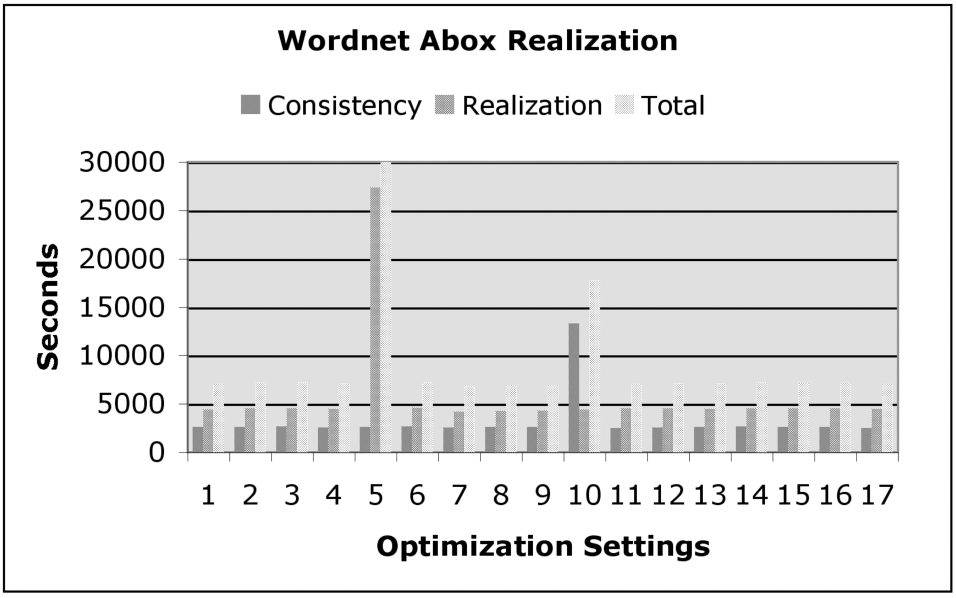
The standard setting (S1) was selected with the goal to ensure a good overall performance. This is generally confirmed by these benchmarks. For S2 some KBs (e.g., WebMin2) have smaller runtimes than in S1. The positive effect can be explained by the low success rate of the completion test (e.g., 0% for WebMin 2, 66% for InfoGlue) and the incompleteness of this technique in case it reported an (possibly unavoidable) inconsistency. For some KBs such as LUBM and UOBM-Lite we notice a slowdown of 10%. S3 shows that the ABox precompletion technique is advantageous for most KBs and even essential for InfoGlue and UOBM-ite. This is due to the reduced overhead in rebuilding initial data structures. S4 indicates that the missing precompletion dominates the increase in runtime. S5 has a very detrimental effect on the runtime. This clearly demonstrates the effectiveness of the individual pseudo model merging technique for ABox realization. S6 is virtually identical to the standard setting except for WebMin 2, where we observe a speed-up of almost 50%. These results indicates that this technique does not seem to be very effective for these KBs. S7 mostly shows the positive effect of the sets-of-individuals-at-a-time technique. It is essential for InfoGlue and UOBM-Lite, which both timed out. LUBM is 3 times slower but VICODI is 30% faster. S8-S9 demonstrate that the disabled sets-of-individuals-at-a-time technique dominates the slowdown. The results for S10 are mixed. Some KBs such as InfoGlue show a performance gain due to reduced overhead while others such as WebMin 1/2 have an increased runtime. S11 shows a similar pattern where WebMin 2 is twice as fast as in the standard setting but others slowed down. S12 and S13 are different from the previous ones because they switch techniques *on* that are disabled by default. The only exception for S12 is WebMin 2, which doubled in speed. All others are in the range of the standard setting.

The next section discusses scenarios where these 2 settings are very favorable. S13 behaves similarly to S12 but WebMin 2 has only a speed-up of 20%. S14 shows also mixed results. InfoGlue slowed down by 25% while VICODI and WebMin 2 increased in speed. So, dependency-based instance retrieval is sometimes favorable because it helps

Tbox Logic	CN	R	A	Abox Logic	Inds	Ind. Ass.	Role Ass.
$\mathcal{L}^-$	84 609	40	85 664	$\mathcal{ELR}^+(\mathcal{D}^-)$	269 684	548 578	304 362

(CN = no. of concept names, R = no. of roles, A = no. of axioms)

Load	Tbox
130	90.44



(Load = load time, Tbox = Tbox classification time)

Set 5 = ind. pseudo model merging off, Set 10 = datatype simplification off

Figure 7: Characteristics of the Wordnet 1.7.1 knowledge base.

to separate "clash culprits" and sometimes it causes unnecessary overhead. S15 shows clearly that binary instance retrieval does not improve the runtimes for realization of the 10 KBs. S16-S17 need to be compared to S3, which also switches off precompletion. LUBM slowed down by 30-50% and UOBM-Lite timed out for S16. In these cases S16 and S17 have a positive effect due to the disabled precompletion. All other results are similar to S3.

Evaluation of Very Large Knowledge Bases The previous section evaluated the presented optimization techniques for instance retrieval mostly on the basis of ABox realization. In this section the evaluation of instance retrieval is continued with very large knowledge bases. The first very large KB is Wordnet, which is evaluated with ABox realization only due to lack of a sufficient number of specific benchmark queries. The other two very large KBs are LUBM and UOBM. They are evaluated using the execution of grounded conjunctive queries, which were designed by the developer of these KBs. Both KBs are tested for ABox size scalability using the standard optimization setting. Furthermore, they are also evaluated against the 17 settings from the previous section.

Wordnet The Wordnet OWL-DL KB consists of three files with a total size of 102MB. Its characteristics are shown in Figure 7. The $\mathcal{L}^-$ TBox consists of 84K concept names and 85K axioms defining a given taxonomy. The ABox logic is $\mathcal{ELR}^+(\mathcal{D}^-)$ and indicates

Knowledge Base	Tbox Logic	Abox Logic
LUBM-Lite	$\mathcal{ELH}$	$\mathcal{ELHR}^+(\mathcal{D}^-)$
LUBM	$\mathcal{ALCH}$	$\mathcal{SH}(\mathcal{D}^-)$

U	Individuals	Ind. Assertions	Role Assertions	Load	Prep
5	102 368	315 139	309 393	90	60
10	207 426	641 822	630 753	196	153
20	437 555	1 356 017	1 332 029	472	456
30	645 954	2 001 556	1 967 308	726	946
40	864 222	2 676 802	2 630 656	990	1 417
50	1 082 818	3 355 749	3 298 813	1 296	1 758

U = no. of universities, Load = load time, Prep = KB preparation time.

Table 7: LUBM Abox characteristics (time in secs).

the use of transitive roles and OWL-DL datatype properties. By analogy to the previous section we evaluated ABox realization using the 17 settings. The results are displayed in the lower part of Figure 7. The runtimes for most settings are in the range for 8 000 seconds and do not vary much. Setting S5 timed out after 30 000 seconds. This result is in line with the lessons learnt from testing large KBs. S10 is the other exception with a runtime increased by a factor of 2.5. This clearly demonstrates the advantage of the datatype property optimization technique. S15-S17 show a speedup of roughly 10% due to the reduced overhead of the disabled techniques.

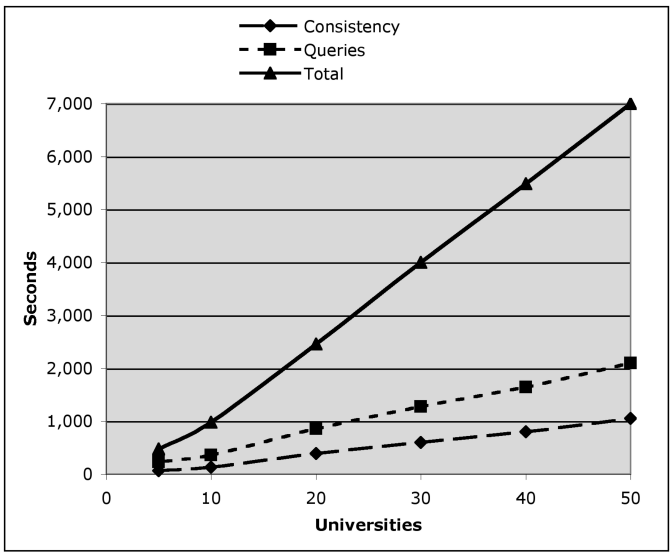
LUBM The LUBM benchmark has the big advantage of being scalable. LUBM was tested with 5-50 university, each with all departments. This results for 50 universities in 1082K individuals, 3355K individual assertions, and 3298K role assertions. Two different TBoxes were used in order to investigate the influence of the GCI absorption technique (see also the discussion in [CGG+07a, Section 2.1.3]. An overview about the characteristics and sizes of the LUBM benchmarks is given in Table 7.

Each benchmark was evaluated with 14 grounded conjunctive queries designed by the authors of LUBM. The benchmark log recorded the runtime for the following phases: (i) loading the input files; (ii) data structure setup for the KB. These runtimes are identical for both TBox variants. The other recorded runtimes are (see the second to fifth columns in Figures 8 and 9): (iii) time for the initial ABox consistency test that precedes the first query execution and initializes appropriate data structures and indexes; (iv) nRQL ABox index generation time; (v) time to execute all 14 queries; (vi) total time consumed by the benchmark. The right part of Figures 8 and 9 shows a graph displaying curves for the runtime of the ABox consistency test (dashed line), query execution (dotted line), and the total benchmark time (solid line).

The graph for LUBM-Lite in Figure 8 gives evidence of RACER's linear scalability for this benchmark type. The total runtime is even dominated by the load and preparation time while ABox consistency and query execution exhibit a straight line with a much smaller gradient than that of the total runtime.

The gradients of the straight lines shown in the graph in Figure 9 are similar for query

U	C	I	Q	T
5	67	39	228	478
10	128	153	363	982
20	391	307	857	2 460
30	600	487	1 274	4 003
40	798	669	1 642	5 487
50	1 055	859	2 100	7 000



U = no. of universities, C = time for initial Abox consistency test,
I = nRQL Abox index generation time, Q = nRQL query execution time,
T = total benchmark time.

Figure 8: LUBM-Lite query runtimes (in secs).

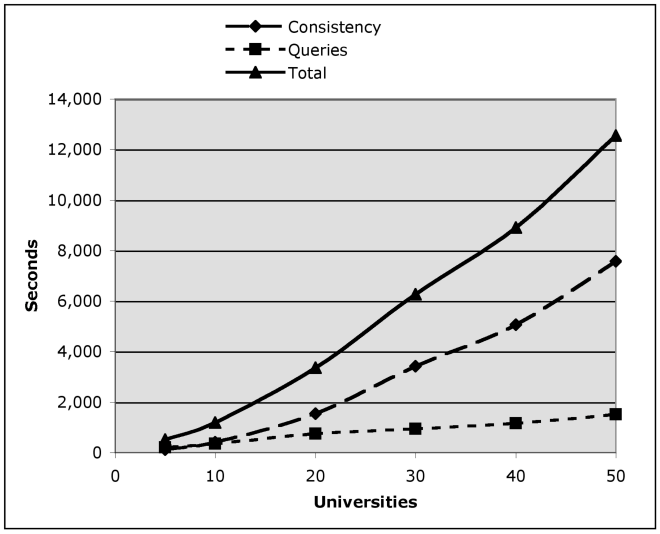
execution but steeper for the ABox consistency test and the total runtime. The more complex TBox causes no penalty for query execution. On the contrary, the query execution is even faster by roughly 30%. The ABox consistency test requires now more than 50% of the total runtime and dominates the benchmark results. This can be explained by the treatment of disjunctions in axioms that were added by the GCI absorption.

We also conducted a second study with LUBM (using both TBox variants) and a selected ABox size of 10 universities. The 14 queries were executed using the 17 settings from the previous section. The Figures 10 and 11 show in the left part the recorded runtimes and in the right part a bar chart illustrating the results (using dark grey for consistency, middle grey for queries, light grey for total runtime). Please note the use of the logarithmic scale in the graph.

The obtained results for LUBM-Lite (see Figure 10) demonstrate that the ABox consistency test is mostly unaffected. Its runtime tripled for setting S10 (datatype simplification switched off) and increased by 30% for S11 (re-use of role assertion switched off). The execution of the queries timed out for S5 (individual pseudo model merging switched off), although ABox realization has not been performed. This emphasizes the importance of this technique also for query-based instance retrieval. The query runtime tripled for S3-S4 and S16-S17, which switch precompletion off. It indicates the effectiveness of the precompletion technique. The other notable slowdown occurred for S12 (factor of 2), which switches on the static index-based instance retrieval. It is obvious that the overhead to build and maintain the index is too high and does not pay off for the execution of the queries.

The runtimes for the ABox consistency test for LUBM (see Figure 11) have tripled compared to LUBM-Lite as expected due to the added disjunctions in the transformed axioms. The recorded runtimes are rather uniform. Setting S10 shows an almost doubled

U	C	I	Q	T
5	118	38	208	510
10	412	81	351	1 183
20	1 542	158	746	3 359
30	3 422	255	934	6 259
40	5 058	326	1 156	8 915
50	7 579	427	1 512	12 541

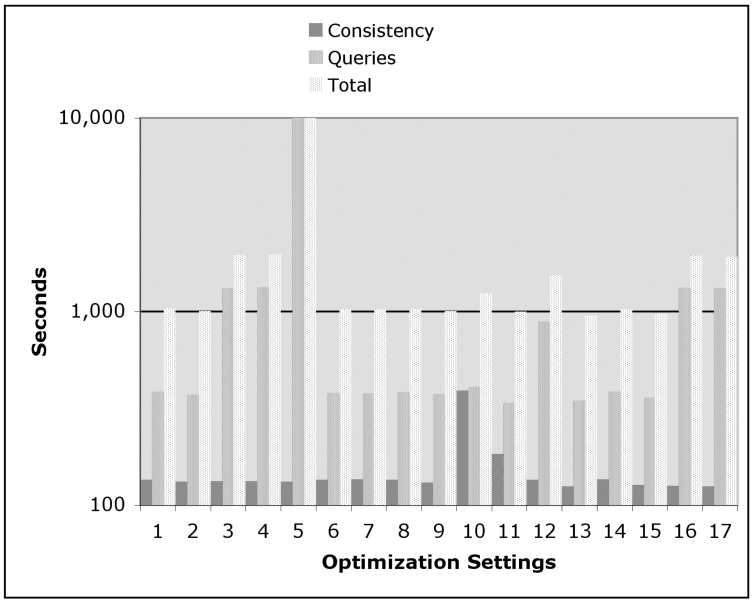


U = no. of universities, C = time for initial Abox consistency test,
I = nRQL Abox generation time, Q = nRQL query execution time,
T = total benchmark time.

Figure 9: LUBM query runtimes (in secs).

runtime and S11 a slight increase. The efficiency of these techniques is compensated by the increased overhead for dealing with disjunctions. Query execution timed out again for S5. Setting S3-S4 and S16-S17, which switch off precompletion, are now a factor 4-5 slower than the standard setting. By analogy to LUBM-Lite S12 demonstrates an increased overhead (factor of 2).

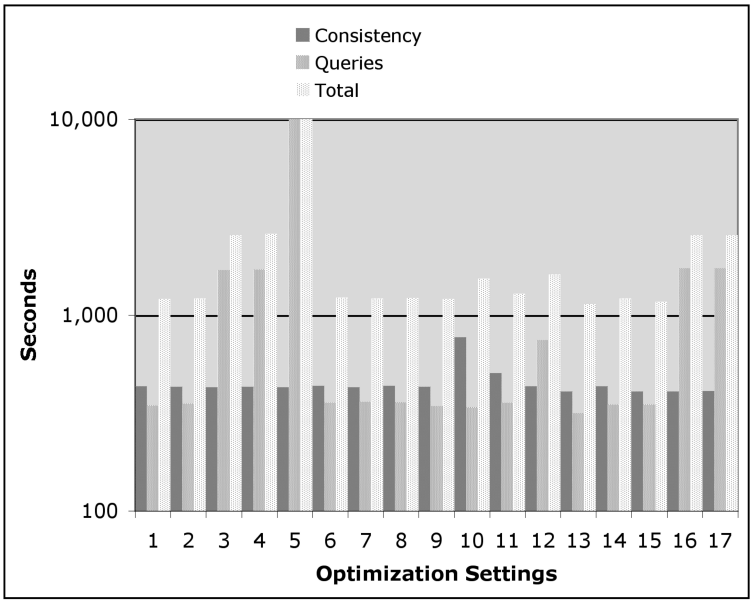
S	C	Q	T
1	135	387	1 039
2	132	371	1 003
3	133	1 326	1 962
4	133	1 340	1 979
5	132	10 000	10 000
6	135	381	1 026
7	136	379	1 033
8	135	385	1 034
9	131	377	1 007
10	391	410	1 247
11	184	339	1 002
12	135	890	1 541
13	125	348	955
14	136	386	1 035
15	127	360	971
16	126	1 331	1 941
17	125	1 322	1 927



S = selected optimization setting, C = time for initial Abox consistency test,
Q = nRQL query execution time, T = total benchmark time.

Figure 10: LUBM-Lite (10 universities) queries (in secs, timeout after 10 000 secs).

S	C	Q	T
1	433	345	1 210
2	431	353	1 216
3	429	1 691	2 556
4	430	1 704	2 598
5	427	10 000	10 000
6	435	357	1 229
7	427	360	1 216
8	434	359	1 224
9	431	343	1 205
10	769	337	1 537
11	504	356	1 284
12	432	744	1 610
13	408	315	1 138
14	433	348	1 214
15	406	349	1 168
16	408	1 736	2 556
17	410	1 730	2 556



S = selected optimization setting, C = time for initial ABox consistency test,
Q = nRQL query execution time, T = total benchmark time.

Figure 11: LUBM (10 universities) queries (in secs, timeout after 10 000 secs).

In [MS06] it was conjectured that a transformation of retrieval queries for description logic ABoxes to disjunctive datalog programs is beneficial in particular if large ABoxes are queried. We reran the tests performed in [MS06] on an AMD 64bit processor under Linux with 4GB main memory. In order to check whether KAON2 is faster than RACER we also ran RACER on this machine (standard settings). As can be seen in Figure 12 the runtimes for answering all 15 LUBM queries are roughly the same for KAON2 and RACER. In Figure 13 the time required for loading the OWL files as well as for setting up index data structures etc. are indicated. RACER requires more time. It performs an ABox consistency test, however, which is not performed by KAON2.

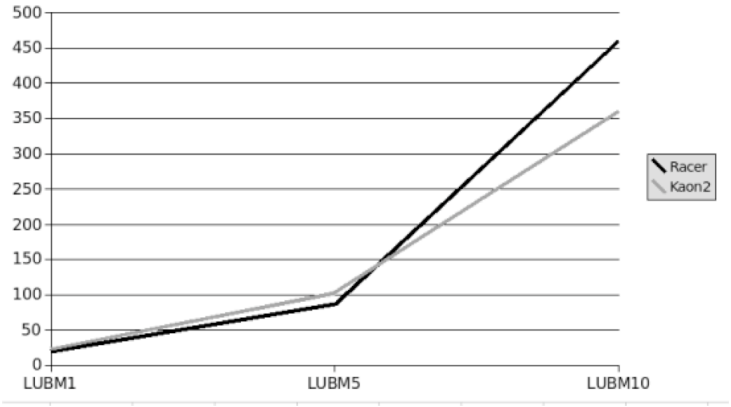


Figure 12: Answering times of KAON2 and RACER for the LUBM queries with a different number of universities.

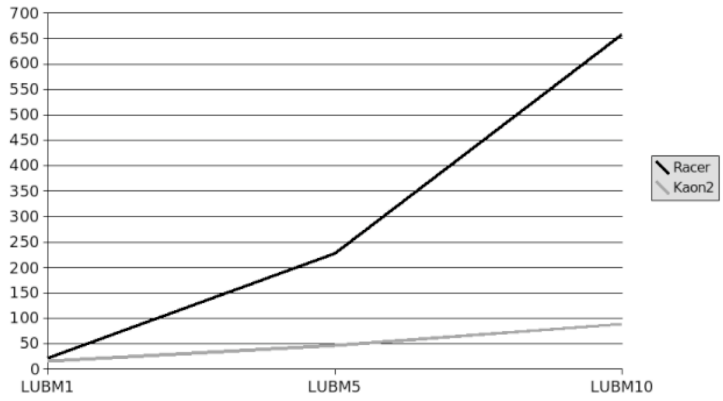


Figure 13: Setup times of KAON2 and RACER for the LUBM queries with a different number of universities.

TBox Logic	CN	R	Axioms	ABox Logic
$\mathcal{ALCF}$	51	49	101	$\mathcal{ALCF}(\mathcal{D}^-)$

(CN = no. of concept names, R = no. of roles)

U	Inds	Ind. Ass.	Role Ass.	L	P	Cons	I	Q	T
1	43 642	116 092	129 695	35	16	100	15	446	608
2	66 900	200 018	222 492	61	35	353	13	1 571	2 041
3	85 055	272 663	302 425	84	59	482	28	3 272	3 920
4	109 919	378 956	419 364	115	111	1 096	31	13 791	15 132
5	138 452	509 902	563 699	160	197	7 670	40	30 000	30 000

U = no. of universities, L = load time, P = KB preparation time,
Cons = time for initial ABox consistency test, I = query index generation time,
Q = nRQL query execution time, T = total benchmark time.

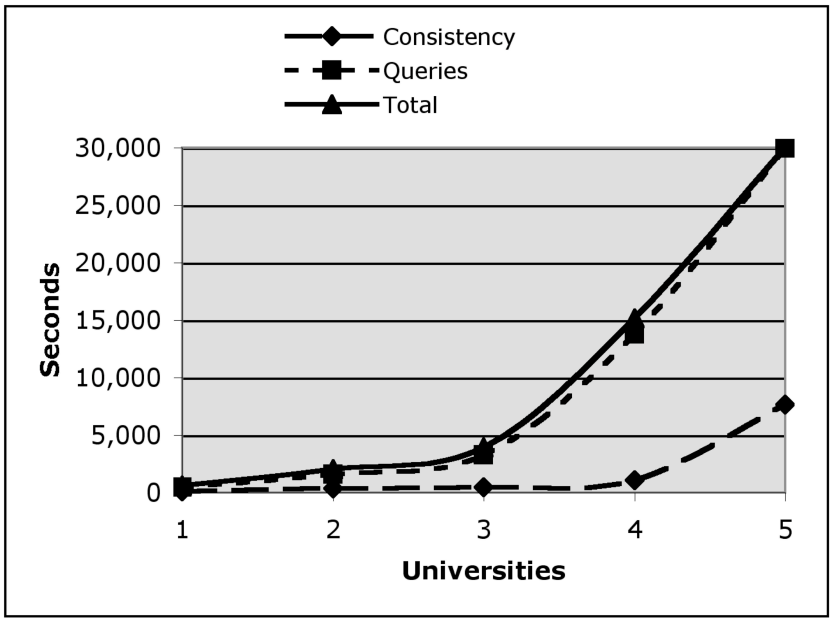


Figure 14: UOBM-Lite benchmark characteristics and runtimes (time in secs, timeout after 30 000 secs).

UOBM The third and last very large KB discussed in this section is the UOBM-Lite benchmark. It is also scalable and was tested with 1-5 universities, each with all departments. The characteristics of the KB and the benchmarks are shown in Figure 14. The logic of UOBM is $\mathcal{ALCF}$ after GCI absorption and the ABox adds datatype properties. The size of the benchmark for 5 universities results in 138K individuals, 509K individual assertions, and 563K role assertions.

Each benchmark was evaluated with 15 grounded conjunctive queries designed by the authors of UOBM. The benchmark has the same structure as for LUBM. The runtimes given in Figure 14 show that RACER's ABox consistency performance scales well for up to 3 universities. The runtime increased by a factor of 2 for 4 universities and a factor of 7 for 5 universities. This degradation of performance is caused by functional roles that enforce the identification of ABox individuals due to non-deterministic restrictions. This type of reasoning is repeated again and again for thousands of individuals. In contrast to LUBM the UOBM benchmark does not allow the unique name assumption. The query execution

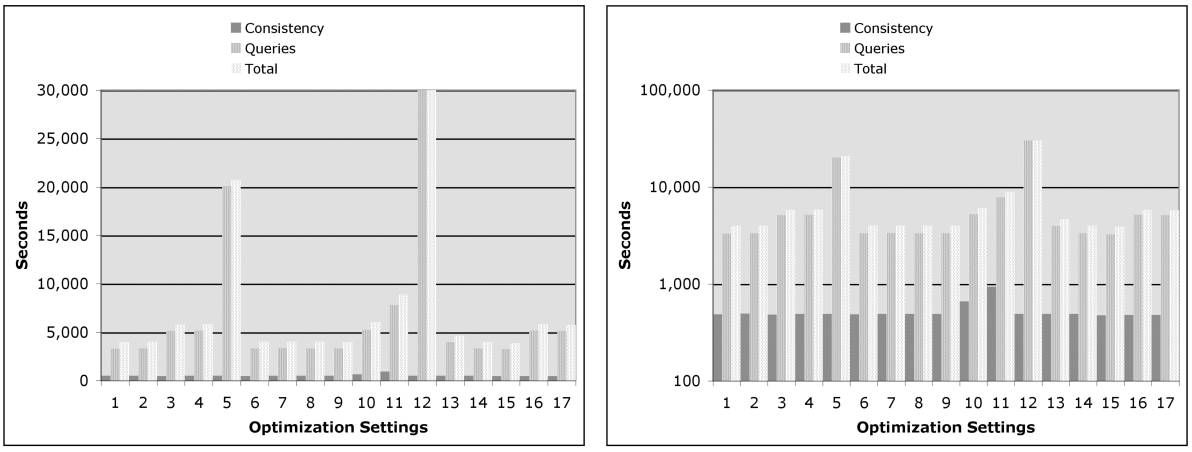


Figure 15: UOBM-Lite (3 universities) queries (in secs, timeout after 30 000 secs).

time also scales well for up to 3 universities. However, for 4 universities it increased by a factor 4 and timed out for 5 universities after 30 000 seconds. The graph in the lower part of Figure 14 displays the curves for the ABox consistency test (dashed line), query execution (dotted line), and the total benchmark time (solid line). The non-linear trend can be easily noticed. It is interesting to remark that 99.86% of the query runtime is spent for 3 of the 15 queries. This performance asks for a refinement of existing or design of new optimization techniques. This is a topic for future work.

For these reasons the second study conducted with UOBM was restricted to a size of 3 universities. We tested the 15 queries using the 17 settings. The results are displayed in Figure 15 where the left bar chart uses a linear and the right one a logarithmic scale (using dark grey for consistency, middle grey for queries, light grey for total runtime). Setting S12, which switches static index-based instance retrieval on, timed out after 30 000 seconds. This result clearly demonstrates that in the case of these 15 queries ABox realization is not worth the effort. S5 switches individual pseudo model merging off and caused an increase of runtime by a factor of 5. Again, this gives evidence for the effectiveness of this technique for instance retrieval without realization. By analogy to LUBM one can notice a slight increase for S3-S4 and S16-S17, which switch off the precompletion, and S10, which disables datatype property simplification. S11 doubled the runtime due to the disabled re-use of role assertions.

5.2 QuOnto

In this subsection we will discuss the experimentation carried out on the QuOnto (Querying Ontologies) tool, a reasoner for the DLs of the *DL-Lite* family [CDGL$^+$07]. We will provide a short description of the tool, and give details on the test setup and the test results.

5.2.1 Tool Description

QuOnto [15] is a free (for non-commercial use) Java-based reasoner for *DL-Lite* with GCIs.

QuOnto is able to manage a large amount of concept and role instances (from thousands to millions) through relational database technology, and implements a query rewriting algorithm for both consistency checking and query answering of complex queries (unions of conjunctive queries) over *DL-Lite* knowledge bases, whose ABox is managed through relational database technology. Currently, it supports its own Java-based interface, and accepts inputs in a proprietary XML format.

TBox Specification in QuOnto As already said, Knowledge Bases (KBs) managed in QuOnto are specified in the DLs of the *DL-Lite* family. DLs of this family are able to capture the main notions of conceptual modeling formalism used in databases and software engineering such as ER and UML class diagrams. Basically, *DL-Lite* assertions allow for specifying (in a controlled way) *ISA* and *disjointness* between concepts and roles, *role-typing*, *participation* and *non-participation constraints* between a concept and a role, *functionality restrictions* on roles, *attributes* on roles and concepts. The DLs of the *DL-Lite* family differ one another for the kind of assertions they allow (among those mentioned above), and for the way in which such assertions can be combined. All such DLs, however, allow for tractable reasoning. Notably, answering unions of conjunctive queries over *DL-Lite* KBs is in LOGSPACE in data complexity, i.e., the complexity measured only w.r.t. the size of the ABox, and the tuning in the use of the assertions in each DL of the *DL-Lite* family is aimed at guaranteeing such a nice computational behavior.

We do not provide here details on the syntax and semantics of the DLs of the *DL-Lite* family, and refer the reader to [CDGL$^+$07, BCG$^+$06, CGG$^+$06, BBC$^+$07] for a in depth and formal description of these matters. We only point out that in QuOnto the TBox is provided in a proprietary XML format.

ABox Specification in QuOnto In QuOnto, the extensional level of the knowledge base is a *DL-Lite ABox*, i.e., a set of plain *membership assertions* [CDGL$^+$07, BCG$^+$06, CGG$^+$06, BBC$^+$07]. For example, for DLs of the *DL-Lite* family that do not allow for the specification of attributes on concepts and roles, an ABox is a set of assertions of the form

$$A(c), R(c, b),$$

where A is an atomic concept, R is an atomic role, c and b are constants. These assertions state respectively that the object denoted by c is an instance of the atomic concept A, and that the pair of objects denoted by (c, b) is an instance of the atomic role R.

One of the distinguishing features of QuOnto is that the ABox is stored under the control of a DBMS, in order to effectively manage objects in the knowledge base by means of an SQL engine. To this aim, QuOnto constructs a relational database which faithfully represents an ABox $\mathcal{A}$: for each atomic concept A, a relational table tab_A of arity 1 is defined, such that $\langle c \rangle \in tab_A$ iff $A(c) \in \mathcal{A}$, and for each role R, a relational table tab_R of

[15] http://www.dis.uniroma1.it/ quonto/

arity 2 is defined, such that $\langle c, b\rangle \in tab_R$ iff $R(c, b) \in \mathcal{A}$ (analogously in the presence of membership assertions involving concept or role attributes).

We point out that the above construction is completely transparent for the user, and that ABoxes in input to QuOnto are simply sets of plain membership assertions, represented in a proprietary XML format.

Query Answering in QuOnto In order to take advantage of the fact that the ABox is managed in secondary storage by a Data Base Management System (DBMS), the query answering algorithm used in the QuOnto system is based on the idea of reformulating the original query into a set of queries that can be directly evaluated by an SQL engine over the ABox. Note that this allows us to take advantage of well established query optimization strategies.

Query reformulation is therefore at the heart of our query answering method. The basic idea of our method is to reformulate the query taking into account the TBox: in particular, given a union of conjunctive queries q over a *DL-Lite* knowledge base $\mathcal{K}$, we compile the assertions of the TBox into the query itself, thus obtaining a new union of conjunctive queries q'. Such a new query q' is then evaluated over the ABox of $\mathcal{K}$, that is over the relational database representing the ABox. Since the size of q' does not depend on the ABox, the data complexity of the whole query answering algorithm is LOGSPACE in data complexity (i.e., the data complexity of evaluating a union of conjunctive queries over a database instance). We refer the reader to [CDGL$^+$07] for more details on the query answering algorithm implemented in QuOnto. We simply point out here that our tool is also equipped with some optimization techniques that aim at "minimizing" each disjunct occurring in the rewritten query q', i.e., each disjunct in the query q' is further rewritten in order to drop some of its atoms to avoid useless join computations.

5.2.2 Test Setup

The main aim of our tests is to show scalability of query answering in QuOnto w.r.t. the growing of the size of the underlying ABox. To this aim, we consider a *DL-Lite* TBox, a set of *DL-Lite* ABoxes of different size, and a set of conjunctive queries posed over the TBox. We then measure the behavior of QuOnto in terms of both the size of the resulting answer sets (i.e., the number of tuples returned by the processing of each query), and the overall time that QuOnto takes to produce these answer sets.

In our experiments, we also compare these time measures with the time measures obtained from evaluating each query directly over each ABox (disregarding the TBox), which, of course, provides a sound but incomplete answer set to each query. This comparison shows that the overhead required by our method w.r.t. simple query evaluation over an ABox is not onerous, and that we can get a complete answer in an efficient way, even on ABoxes of big size.

We finally point out that possible usage scenarios for the reasoning task and the technique (ABox query answering) that we test in these experiments are the online usage scenarios for ABox Access described in the TONES Deliverable D14 [CGG$^+$07b].

All experiments have been carried out on an Intel Pentium IV Dual Core machine, with 3 GHz processor clock frequency, equipped with 1 Gb of RAM, under the operating

system Windows XP professional.

In the following, we provide more details on our test setting.

Ontology TBox To perform our experiments, we considered the OWL Lehigh University Benchmark (LUBM) [16]. LUBM consists of an OWL ontology for modeling universities; i.e., the ontology contains concepts for persons, students, professors, publications, courses, etc., as well as appropriate relationships for such a universe of discourse (see also the brief description provided in the TONES Deliverable 14 [CGG+07b]).

In fact, in our experiments, we considered an approximation of the OWL LUBM TBox that is expressed in *DL-Lite*$_\mathcal{A}$, a DL of the the *DL-Lite* family that has as distinguishing features the ability of specifying attributes on concepts and roles, and ISA on such attributes, and also, analogously to other DLs of the *DL-Lite* family, allows for specifying functionalities on roles (and on the inverse of roles), ISA between concepts and roles (with some suitable limitations), participation and non-participation constraints, etc. [CGG+06].

Notice that we also enriched the resulting *DL-Lite*$_\mathcal{A}$ TBox by adding some TBox assertions to capture particular aspects of the domain that were not caught by the original TBox. For example, we added the role $hasExam$, to model also the courses for which a student has passed the exam. Also, we introduced role attributes, which cannot be expressed in OWL. For instance, we added the role attributes $eventYear$, $examYear$, and $degreeYear$, to allow the specification of the year in which, respectively, an event occurred, a student passed an exam, and a student took a degree. Note that we also imposed that years of both an exam and a degree are years in which some event occurred. This is expressed by means of the following TBox assertions:

$$examYear \sqsubseteq eventYear$$
$$degreeYear \sqsubseteq eventYear$$

It is worth noting that the TBox that we consider in our experiments presents some forms of cyclic dependencies, as shown the following subset of TBox assertions:

$$\begin{array}{ll} Student \sqsubseteq \exists takesCourse & \exists takesCourse^- \sqsubseteq Course \\ Course \sqsubseteq \exists teacherOf^- & \exists teacherOf \sqsubseteq Faculty \\ Faculty \sqsubseteq \exists worksForUniv & \exists worksForUniv^- \sqsubseteq University \\ University \sqsubseteq \exists hasAlumnus & \exists hasAlumnus^- \sqsubseteq Student \end{array}$$

It is possible to show that, given an ABox $\mathcal{A}$, there exists no finite first-order structure $\mathcal{S}$ such that, for every conjunctive query q, the set of answers to q over the knowledge base constituted by the above TBox and the ABox $\mathcal{A}$, is the result of evaluating q over $\mathcal{S}$. This property demonstrates that answering queries in *DL-Lite* goes beyond both propositional logic and relational databases.

Ontology ABox Rather than using the benchmark generator available for the LUBM ontology, which generates synthetic extensional data corresponding to the LUBM ontology, we considered ABoxes constructed from *real data* concerning the university domain.

[16] `http://swat.cse.lehigh.edu/projects/lubm/`

Name	ABox size (number of assertions)	Data description
$ABox_1$	118075	Before 1995 (concerning students living in Rome)
$ABox_2$	165049	Before 1995
$Abox_3$	202305	Before 1997 (concerning students living in Rome)
$ABox_4$	280578	Before 1997
$ABox_5$	328256	Before 1999 (concerning students living in Rome)
$ABox_6$	482043	Before 1999

Table 8: ABoxes used for tests

These data are taken from the information systems of the Faculty of Engineering of the University of Rome "La Sapienza", and refers to the period 1990-1999. Starting from these data, we constructed six different ABoxes of growing size. These are presented in Table 8 (before 199x means that the ABox concerns only data from 1990 to 1995).

Queries In order to show the benefits of using QuOnto, we consider the following queries:

Query 1 : It asks for all persons living in Rome that obtained at least a '30' as exam mark:

$$q(x) : -Person(x), address(x,' ROMA'), examRating(x, y,' 30').$$

Query 2 : It asks for the names of all students that took a course, together with the name of such a course:

$$q(z, w) : -Student(x), name(x, z), takesCourse(x, y), name(y, w).$$

Query 3 : It asks for the names of all persons that passed at least an exam:

$$q(x) : -Person(x), hasExam(x, y).$$

Query 4 : It asks for the names of all persons whose address is the same as the address of the place for which their advisor works:

$$q(z) : -Person(y), name(y, z), address(y, w), advisor(y, x), worksFor(x, v), address(v, w).$$

Query 5 : It asks for all students that took a course together with the address of the organization for which the course teacher works:

$$q(x, c) : -Student(x), takesCourse(x, y), teacherOf(z, y), worksFor(z, w), address(w, c).$$

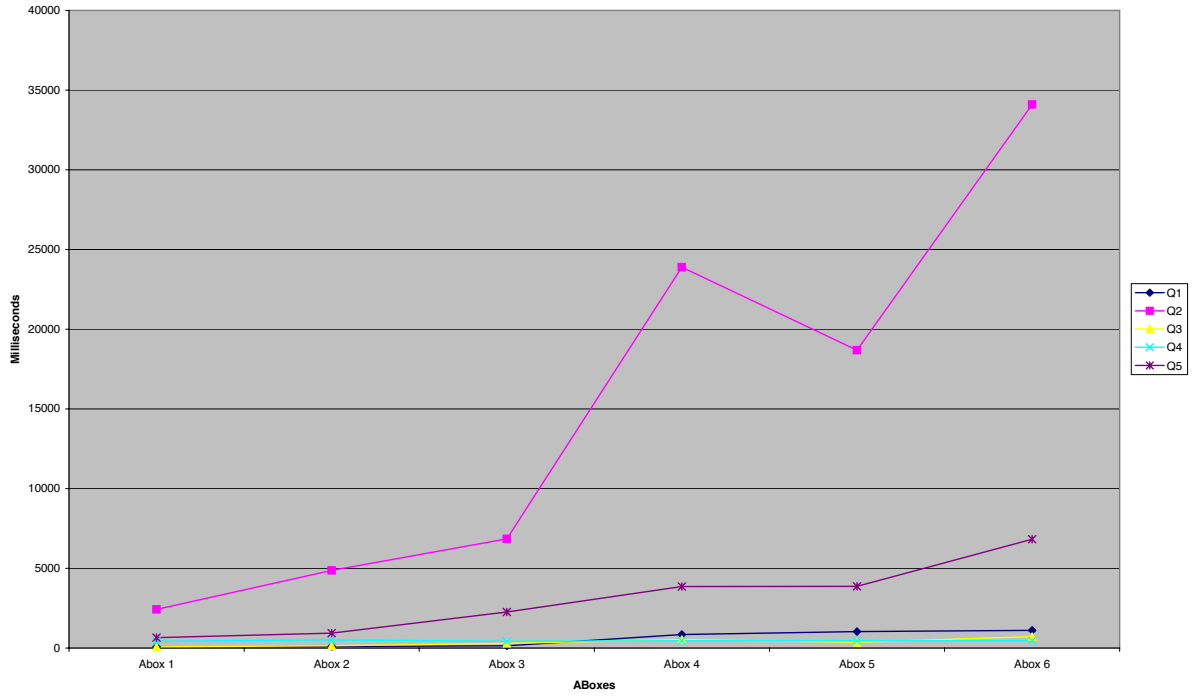


Figure 16: QuOnto execution time for query answering

5.2.3 Test Results

The main results of our experiments are given in Figure 16 and Figure 17, which respectively show the performance (execution time) for answering each query w.r.t. the growth of the size of the ABox, and the number of tuples returned by each query.

To make the results more readable, we provide below also a table providing exact times for query answering in QuOnto (values are in milliseconds).

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Abox 1	78	2422	78	422	657
Abox 2	94	4875	140	516	938
Abox 3	140	6844	296	421	2266
Abox 4	844	23891	532	454	3860
Abox 5	1031	18687	359	453	3875
Abox 6	1110	34094	18	453	6828

For each query, the execution time comprises the time needed for rewriting the input query, minimizing it, and evaluating it over the ABox. We point out that the time needed for query rewriting and query minimization is negligible w.r.t. the overall execution time, and that the major time consuming process is the evaluation of the rewritten query over the ABox (for which we have always increasing values at the growth of the underlying ABox). This depends both on the number of disjuncts occurring in the rewritten query (which is a union of conjunctive queries), and the number of membership assertions of the ABox involving concepts, roles, and attributes occurring as predicates of the query

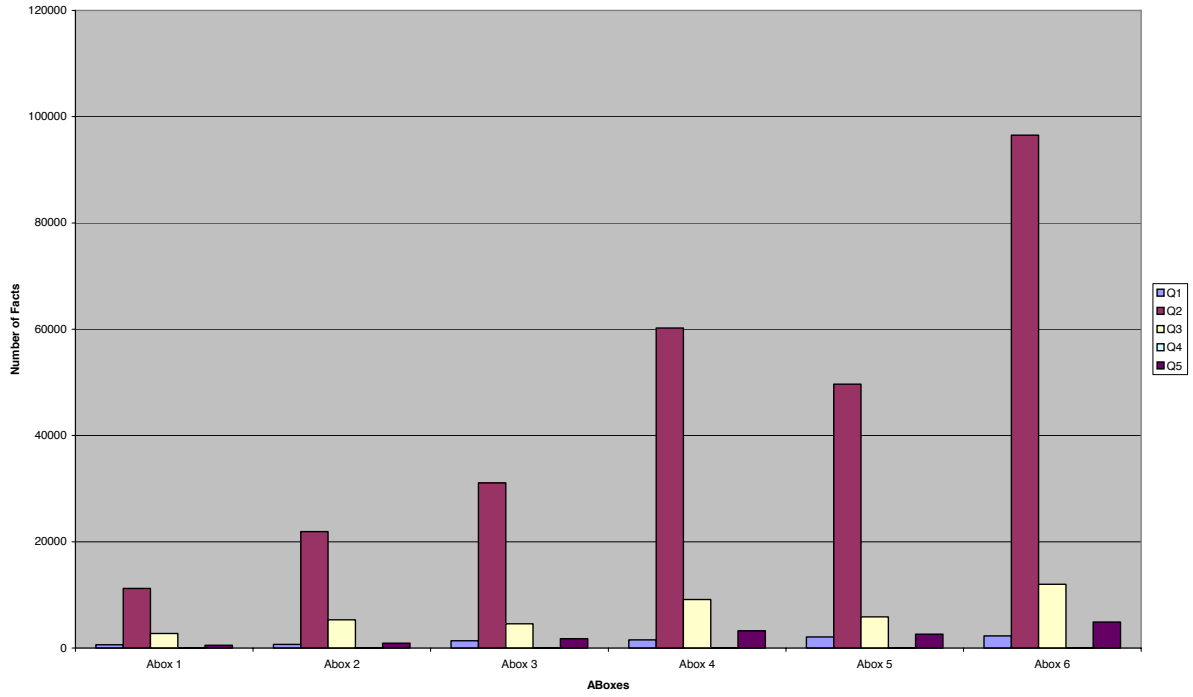


Figure 17: number of tuples returned by QuOnto for each query

atoms. As an example, we provide below the rewriting of the query Q2 (expressed in Datalog notation), for which we measure the highest execution times for each underlying ABox.

$$
\begin{array}{rcl}
q(Z,W) & :- & name(Y,W), examRating(X,Y,n_0), name(X,Z) \\
q(Z,Z) & :- & takesGraduateCourse(X,X), name(X,Z) \\
q(Z,W) & :- & name(Y,W), takesGraduateCourse(X,Y), name(X,Z) \\
q(Z,W) & :- & name(Y,W), hasExam(X,Y), name(X,Z) \\
q(Z,Z) & :- & examRating(X,X,n_0), name(X,Z) \\
q(Z,Z) & :- & takesCourse(X,X), name(X,Z) \\
q(Z,Z) & :- & hasExam(X,X), name(X,Z) \\
q(Z,W) & :- & name(Y,W), takesCourse(X,Y), name(X,Z)
\end{array}
$$

We notice that QuOnto shows good scalability w.r.t. the growth of the size of the ABox, and that execution times are always small, even for answering queries that are rewritten in union of conjunctive queries with several disjuncts (e.g., the rewriting of query Q5 contains around 40 disjuncts). The ability of QuOnto to provide efficient query answering is made also evident by the fact that the overhead required by the QuOnto query answering strategy w.r.t. simple query evaluation over the ABox (that can be seen as a flat relational database) is not onerous, i.e., for query answering in QuOnto we get results comparable to standard query evaluation over relational databases. The table below provides time measures obtained from the evaluation of each of our test query directly over each of our ABoxes, disregarding the TBox (values are in milliseconds).

	Q1	Q2	Q3	Q4	Q5
Abox 1	1	1047	1	1	171
Abox 2	62	2438	1	94	343
Abox 3	31	3235	1	79	875
Abox 4	47	6890	2	46	1546
Abox 5	31	5704	2	63	1562
Abox 6	31	8640	2	63	2844

6 Query Formulation Support

While designing a tool, its *usability* evaluation is an essential step of a User Centred Design Methodology, together with the identification of users and of their needs, the correct exploitation of this information to develop a system that, through a suitable interface, meets the users' needs, and, obviously, the system usability evaluation. In fact, by analysing the experiment results, we can improve the interaction between the users and the system.

Several different definitions of usability exist; we adopt from the ISO 9000 the following very comprehensive definition of usability: "the extent to which a product can be used with efficiency, effectiveness and satisfaction by specific users to achieve specific goals in specific environment". From this point of view, the usability is the quality of interaction between the user and the overall system. It can be evaluated by assessing three factors:

- effectiveness, i.e., the extent to which the intended goals of the system can be achieved;
- efficiency, i.e., the time, the money, the mental effort spent to achieve these goals;
- satisfaction, i.e., how much the users feel themselves comfortable using the system.

It is worth noting that the usability depends on the overall system, i.e., the context, which consists of the types of users, the characteristics of the tasks, the equipment (hardware, software, and materials), and the physical and organisational (e.g., the working practises) environment. Usability is an essential quality of Software Systems.

An important element in any usability evaluation is the classification of the differences among the skills of evaluators (i.e., the persons evaluating the usability of an interaction design); in particular we deal with two main criteria, **Expert-based** criteria and **User-based** criteria. In the former, experts are requested to evaluate a prototype, comparing it w.r.t. existing rules and guidelines; in the latter evaluators assess usability through real users, having them *using* a prototype. User-based criteria include, among others, Observational evaluation method, Survey evaluation method, and Controlled experiment method.

The *observational evaluation method* involves real users that are observed when performing tasks with the system (depending on the stage of the project, what "the system is" ranges from paper mock-ups to the real product). This method offers a broad evaluation of usability. Depending on the specific situation, we may either apply the Observational

Evaluation by direct observation or record the interaction between the users and the system (using Usability Lab). Recording (done by video camera) is more valuable, since it allows storing a lot of information, for example the critical points during the interaction (when the user has to consult the manual, when and where s/he is blocked, etc.), the time a user spends to perform a task, the mistakes a user makes, and so on. Obviously, recording, using camera, is much expensive (especially for the time required to analyse the recorded data). Various protocols are possible while observing users:

- The *think aloud protocol* provides the evaluator with information about cognitions and emotions of a user while the user performs a task or solves a problem. The user is instructed to articulate what s/he thinks and feels while working with a prototype. The utterances are recorded either using paper and pencil or using audio and/or video recording. By using the Think Aloud Protocol, the evaluator obtains information about the whole user interface. This protocol is oriented towards the investigation of the user's problems and decisions while working with the system.

- *Verbal protocols* aim at eliciting the user's (subjective) opinions. Examples are interviews and questionnaires. The difference between oral interview techniques and questionnaire based techniques lies mainly in the effort for setup, evaluating the data, and the standardisation of the procedure.

In the *survey evaluation method*, structured questionnaires and/or interviews are used to get feedback from the users. This method offers a broad evaluation of usability since from the user's viewpoint it is possible to identify the critical aspects in user-system.

The *controlled experiment method* is particularly valid to test how a change in the design project could affect the overall usability. It may be applied in any phase during the development of a system; it provides more advantages when it is possible to test separately the alternative designs, independently by the whole system. This method mainly aims at checking some specific cause-effect relations, and this is possible by controlling as many variables as we can.

The usability evaluation we have carried on is composed by the following steps:

- **User Analysis**. A user classification method identifies a certain number of features, which permits the labelling of a homogeneous group of users. The number and the kinds of groups differ depending on the specific classification. However, there is at least a general agreement on the initial splitting of the users into two large groups: those who have had a certain instruction period and have technical knowledge, and those who do not have specific training in computer science. Actually, in the experiment we call those two groups skilled and unskilled users respectively. The several features roughly characterise the unskilled user: s/he interacts with the computer only occasionally, s/he has little, if any, training on computer usage, s/he has low tolerance for technical aspects, s/he is unfamiliar with the details of the internal organisation of an information system. Usually, this user does not want to spend extra time in order to learn how to interact with a system, and finds it irritating to have to switch media, e.g., to manuals, in order to learn how to interact with the system. Moreover, the unskilled user wants to know where s/he is and

what to do at any given moment of the interaction with the system. Notice that the unskilled user is very similar to Cuff's casual users. On the other hand, skilled users possess knowledge of considered systems, information systems, etc., and often like to acquire a deep understanding of the system they are using.

- **Experiment Design**. The main goal of the experiment design is to propose a complexity model and to validate the metrics used to measure the system usability. In order to estimate the usability, the evaluators provide not only to define precisely what one is going to watch/measure, but also they provide to develop tasks for users to perform; moreover, they measure relevant parameters (metrics) of user performance, and they validate values collected during the experiments.

- **User Teaching**. The goal of the usability evaluation experiments is to measure the effectiveness and the efficiency of the system and the user's satisfaction using it, discarding each aspect involving the learning time of the different environments. For this reason, this step aims at making users aware of system functionality and experiments modalities. Following this guideline during the teaching users step, we set up exhaustive explanation about each tool. In this way the users were totally acquainted with the usage of the system and, during the final test, they were free of concentrating exclusively on the tasks execution.

- **Experiment Execution**. During this step, the evaluators provide to explain users about the experiment and to assign users the developed task. Moreover, they take notes of any conditions or events, which occur during the experiment.

- **Usability Analysis**. The evaluators collect the information on each performed test and in order to obtain statistically significant metric values, is very important validate such results with an Anova test (AN analysis Of Variance test). While the analysis of results is in charge of the evaluators, all people involved in the experiment, as mentioned in the User Centred Design Methodology perform the evaluation of the usability.

6.1 QueryTool

We recall here very briefly the man idea behind the query tool. Details can be found in previous deliverables.

The query tool is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query (or a non nested Select-Project-Join SQL query) ready to be executed by some evaluation engine associated to the information system.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's rich vocabulary. The user can exploit the ontology's vocabulary to formulate the query, and

she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology through intensional navigation. The user may specify her/his request using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, giving instead an explicit meaning to a query and to its sub-parts through classification.

Query expressions are compositional, and their logical structure is not flat but tree shaped; i.e. a node with an arbitrary number of branches connecting to other nodes. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases. The latter can contain nested noun phrases themselves.

The focus paradigm is central to the interface user experience: manipulation of the query is always restricted to a well defined, and visually delimited, sub-part of the whole query (the *focus*). The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system pro-actively suggests only the operations which are consistent with the current query expression; in the sense that do not cause the query to be unsatisfiable. This is verified against the formal model describing the data sources.

6.2 Test Setup

The method we use for the experiments is the observational evaluation method and, in particular, the Think Aloud and Verbal Protocols. Also, we record the tests with a video camera in order to valuate rigorously a lot of information, for example the critical points during the interaction (when the user has to consult the manual, when and where s/he is blocked, etc.), the time a user spends to perform a task, the mistakes a user makes, and so on.

6.2.1 The domain and the users

The evaluation of the query tool presented in this chapter is mainly a follow-up to the outcome of the European IST RTD project SEmantic Webs and AgentS in Integrated Economies (SeWAsIE, IST-2001-34825), in which a first version of the query tool was developed. For this reason, the applicative scenario and the ontology are derived from the SeWAsIE project, which was about supporting the textile industry . The project's ontology used in this evaluation is written in the description logic $\mathcal{ALCQI}$ and it comprises around 300 concepts and 70 roles. Since in this case we are not evaluating the ontology itself, but the user interface of the query tool, the details of the involved ontology are not relevant here. On the other hand, it is important that we have evaluated the query tool

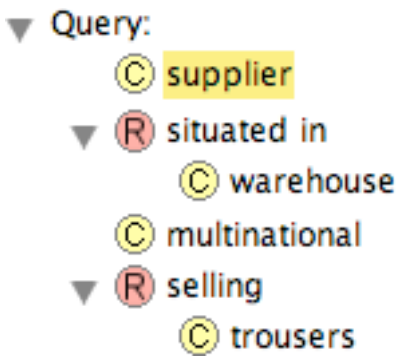


Figure 18: Sample query

in a real world scenario, that we know well due to our past research projects.

Three people are involved in this session of the usability evaluation experiment. In particular, while two people are very skilled in computer science, the other one is unskilled in computer science and he uses the computer only at work. These users belong to the employees of provincial and municipal offices class, and they well represent the end-users for the Query tool environments. In the SeWAsIE scenario, provincial and municipal offices work together with the textile industries giving trade-union help, economic strategies and other services required. These employees are constantly kept updated about all relevant news and regulations, and they work in a flexible structure to provide services and opportunities; their main activity is to provides services to craftsmen and small businesses; their main services are: personnel administration; management consulting; management training.

We consider the end-users as Domain Expert users, differently from the five students that perform the complexity model experiment session, that we classified as Non Domain Expert (NDE). This classification in NDE and DE is very important in our context; in fact, the main goal of our experiment is to demonstrate the easy of use of the Query tool independently from the previous deep knowledge of the domain.

6.2.2 Designing experiments

The objective of our study is to measure and understand the use complexity of the Query tool. More specifically, we are interested in determining how much is difficult for the users to construct queries, and to understand its results. In order to evaluate which is the quality of the interaction between the domain expertise of the users and the query paradigm used in the Query tool environment to construct queries, we develop different tasks for the users (the query writing and query reading tasks); moreover, we design a model of complexity, a number of query of increasing complexity, and a questionnaire to capture relevant aspects of the interface interaction.

In the model of complexity, to each query we assign a complexity tree: nodes are associated to concepts of the query and are identified by their relative order within a level of the tree, and edges are associated to the property/compatible relation among concepts in the query with weight c_l^n, where $c_l^n = 0.1$ if there is a *property* relation and $c_l^n = 0.2$ if there is a *compatible* relation; the tree has depth $l_{\max}$ and each level l has n_l total nodes.

Query_i	Num Level	Num Node	Avr_num suc_per_node	Avr_num node_per_level	Complexity
Query_1	2	2	1,00	1,00	0,30
Query_2	3	3	1,00	1,00	0,60
Query_3	2	3	2,00	1,50	0,80
Query_4	3	4	1,25	1,33	0,80
Query_5	5	8	1,39	1,60	2,45
Query_6	6	9	1,20	1,50	2,80
Query_7	6	11	1,50	1,83	4,00
Query_8	7	12	1,42	1,71	5,60
Query_9	8	11	1,39	1,38	6,05

Low Complexity

Medium Complexity

High Complexity

Figure 19: Test queries complexity

Starting from the query tree, we define a function to calculate the complexity of the query, expressed with the following formula:

$$\sum_{l=1}^{l_{\max}} l \cdot \left(\sum_{n=1}^{n_l} c_l^n \cdot n \right) \cdot \frac{n_{l+1}}{n_l} \qquad \text{(with } c_1^1 = 0.1\text{, and if } n_{l+1} \text{ is undefined then } n_{l+1} = n_l\text{)}$$

For example, the query "Tell me the suppliers situated in a warehouse, and which are multinationals selling trousers", shown in Figure 18, has the following complexity:

$$1 \cdot (0.1 \cdot 1) \cdot 2 + 2 \cdot (0.2 \cdot 1 + 0.1 \cdot 2) \cdot 0.5 + 3 \cdot (0.1 \cdot 1) \cdot 1 = 0.9$$

Using this model, we devised a list of queries with increasing complexity; the values of increasing complexity are showed in the Figure 19, where we highlight several characteristics of the queries (e.g. number of levels, the average number of successors per node, etc.).

The values of the metrics we use to describe the performance for the usability evaluation are: the time spent to compose the query, the number of steps used to compose the queries, the number of focus change, the number of mistakes, the number of cancellations, and the number of clicks on the Query Manipulation Pane.

In order to validate the complexity model, the performance metrics, the queries, and the questionnaire, we have done a preliminary session of the experiment performed with the Non Domain Expert users. It is interesting to note that the query complexity (pink line of the Figure 20) has the same increasing behaviour of the metrics observed during this preliminary test session. By inspecting the time spent values in the Figure makes clear that the users have learnt the system by using it; in fact, for the queries of high complexity (Query-7, 8, 9), the function of the time spent metric increases more slowly than the other metrics.

The proper experiment session involved the three Domain Expert end-users above mentioned. Each user has a workstation.

This test is composed by two sub-sessions: the first for the two skilled people and the second for the unskilled user. For each sub-session, we design two tasks set to be performed by the users:

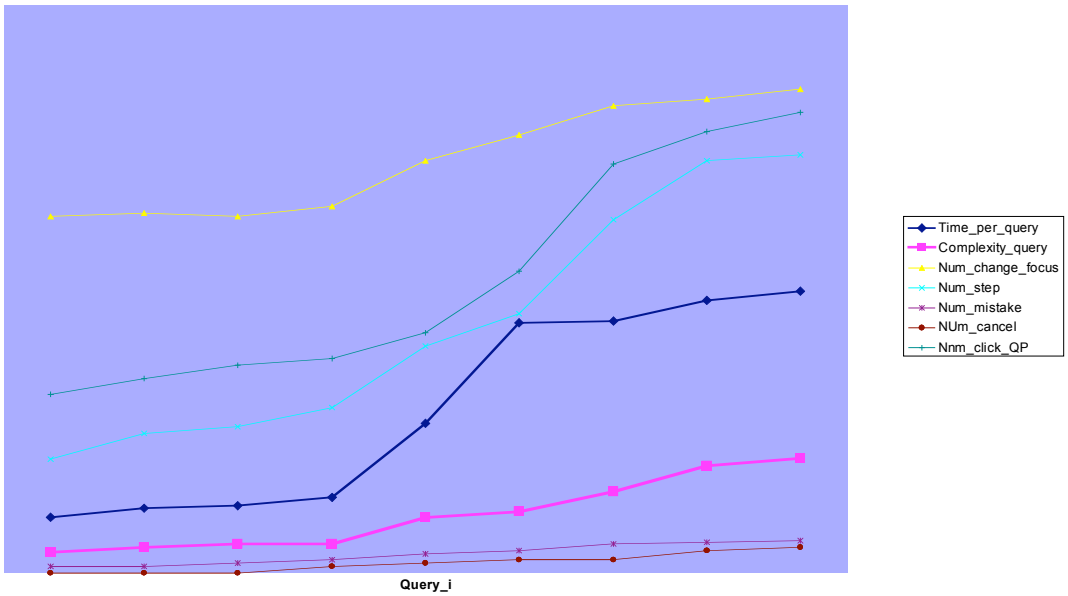


Figure 20: Complexity vs average values of metrics

- *skilled users tasks:* to compose different queries (Query-1, 4, 9 for the User-1; Query-2, 5, 8 for the User-2), and to read the results of the assigned queries, analysing them;
- *unskilled users tasks:* to propose thinking the queries that the user composes usually, constructing them using the Query tool, and to read the results of these queries, analysing them.

Before performing the tests, we set up the training session, instructing users about the tasks to perform. During the performance, we observe the users and collect the measures of the metrics defined above. In this phase, we ask the users to think aloud, by describing their intentions, expectations, and their problems. In particular:

- the engineers instruct the skilled users about modalities of the experiment and they introduce the main goal of the Query tool without describing the functionalities of the tool;
- the users interact with the tool to understand how it works. During this auto-training session, the engineers record with a camera relevant performances. After that, each subject was presented with tasks;
- while the first user perform the tasks, the engineers observe the session of test, and they record the users' utterance using camera (Think Aloud Protocol);
- the engineers propose to the users the designed questionnaire and users make it. The engineers collect questionnaires;
- while the second user perform the tasks, the engineers observe the session of test, and they record the users' utterance using camera (Think Aloud Protocol);

Query_i	Num Level	Num Node	Avr_num suc_per_node	Avr_num node_per_level	Complexity		
Query_1	2	2	1,00	1,00	0,30		Low Complexity
Query_4	3	4	1,25	1,33	0,80		Medium Complexity
Query_9	8	11	1,39	1,38	6,05		High Complexity
Query_2	3	3	1,00	1,00	0,60		
Query_5	5	8	1,39	1,60	2,45		User_1 skilled
Query_8	7	12	1,42	1,71	5,60		User_2 skilled
Query_10	3	6	2,13	2,00	1,20		User_3 unskilled
Query_11	4	6	1,44	1,50	1,70		

Figure 21: Complexity of the performed queries

		Time spent	Num change focus	Num step	Num mistake	Num cancellation	Num QMP click
Utente_1	Query_1	0,40	0	2	0	0	0
Utente_1	Query_4	1,17	0	3	0	0	1
Utente_1	Query_9	12,20	1	10	1	1	2
Utente_2	Query_2	1,30	0	2	0	0	2
Utente_2	Query_5	9,00	1	4	1	1	2
Utente_2	Query_8	14,00	1	9	2	1	3
Utente_3	Query_10	8,30	0	4	0	0	2
Utente_3	Query_11	10,15	0	6	1	1	4

Figure 22: Values of the metrics of the performed queries for each User

- the engineers propose to the users the designed questionnaire and users make it. The engineers collect questionnaires;
- the engineers teach the unskilled user about modalities of the experiment, and they describe the user the Query tool functionalities. After this training session, the subject was presented with Query tool tasks;
- while the user perform the tasks, the engineers observe the session of test, and they record the users' utterance using a camera (Think Aloud Protocol);
- the engineers propose to the user the questionnaire and the users compose it. The engineers collect the questionnaire.

6.3 Test Results

We calculated the complexity of queries defined by the unskilled users, in particular, in the Figure 21, we highlight some characteristics of the queries performed in this section of the experiments

For the queries performed by the end users, we show the values of the metrics describing the performance for the usability evaluation (see Figure 22). These measures are calculated using the video recorded during the experiments session.

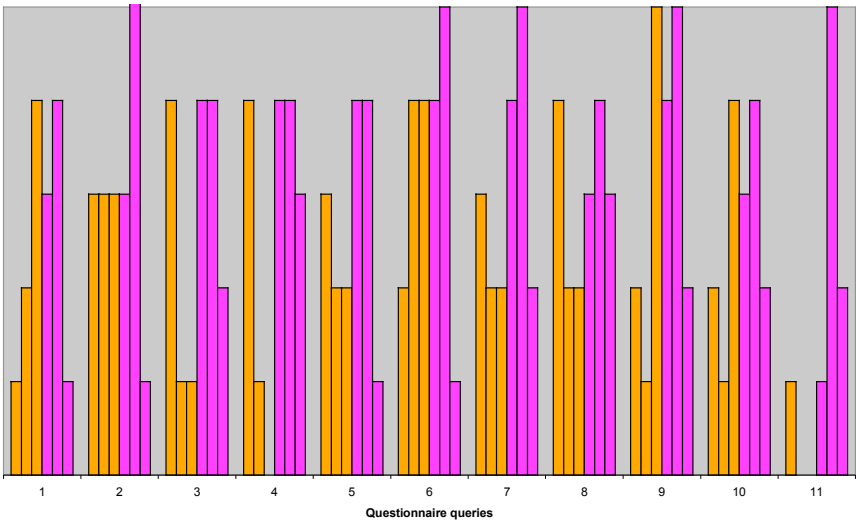


Figure 23: Histogram describing the results of the questionnaire

The auto-training session of the skilled users turned out some usability aspects, in particular it is not very clear: the conceptual difference between Add a compatible term button and Add a propriety button (see the Figure 21); why some proprieties are present two or more times in the Add a propriety list; why it is impossible clicking freely on the tab that represent the main activities to construct the query; why the system takes a long time to answer a query.

Different observations come up from the unskilled user that asked for: biggest fonts for the natural language query representation (in the text box); a method to compact the query manipulation pane; the possibility to customise the values in the add concept list.

The Figure 23 shows the histogram containing the results of the questionnaire assigned to the end users, (Non-domain expert users vs Domain expert users). In particular, we use a colour code to identify the Non Domain Expert and the Domain Expert users. We calculated the histogram in order to understand the relationship between the users' satisfaction and their domain experience.

Nevertheless, the observations of the skilled users, done during the auto-training session, these users after the brief training are able to perform the requested writing and reading tasks. In the case of unskilled user, easily, he proposes two valid queries, and the values of time spent to build these queries is relatively low (see Figure 22); moreover, the number of mistakes is irrelevant. Therefore, we conclude that the overall functionality and philosophy of the Query tool interface are well understood by all users.

Moreover, we highlight that the value of time spent to construct queries is independent from the domain expertise of users. In fact, this performance measure is only function of queries complexity. In order to demonstrate that, we calculated the average values of time-spent to construct the low complexity queries, the medium complexity queries, and the high complexity queries for the two classes of users (Non Domain Expert = NDE, and Domain Expert = DE), collected in the Figure 24, validating such results with an Anova test.

Finally, it is worth noting that the questionnaire highlights that the user satisfaction

		Low	Medium	Hight
NDE		1,06	9,31	13,56
DE		1,05	8,87	13,10

Figure 24: Average values of the users time-spent for each class of users

to achieve the specific writing tasks is independent of the user domain experience; in fact observing the histogram in the Figure 24, we note that there are non significant gap between the values representing the average of result values of the Non Domain Expert users (orange col-or) and the same values of the Domain Expert users (pink col-or).

In our context, this aspect is a very strong point, because it demonstrates that the system can be used independently of the user domain expertise, in others words each class of user is able to construct queries using the interface of the Query tool.

6.4 Final considerations

The main goal of our experiment was to demonstrate the easy of use of the Query tool independently of the domain user experience. Wee used the observational evaluation method and, in particular, the Think Aloud and Verbal Protocols. We described the evaluation experiments adopting a general user-based criteria schema.

The designed aspects (e.g., the model complexity) have been validated by a preliminary session of experiment performed with the non-domain expert users (five students). In particular, the test session highlighted that the query complexity has the same increasing behaviour of the metrics and that the users learnt the system using it. Moreover, these results validated the query complexity mode and the questionnaire. Considered the positive results, we have performed the usability experiments session, starting from the training session, instructing users about the tasks to perform, and observing them in order to collect the required figures.

Concluding, the users were able to perform the requested writing and reading tasks. Therefore, we have concluded that the overall functionality and philosophy of the Query tool interface was well understood by all users. Moreover, we have observed that the value of time spent to construct queries is independent of the domain expertise of users, validating such results with an Anova test. Finally, the questionnaires have highlighted that the user satisfaction to achieve the specific writing tasks is independent of the user domain experience; this aspect is a very strong point, because it demonstrates that the system can be used independently of the user domain expertise, confirming that the Query tool system is usable by both end users (domain-expert users) and non-domain expert users.

7 Information Extraction via Abduction

In this section we present a framework for media interpretation that leverages low-level information extraction to a higher level of abstraction and, therefore, enables the automatic annotation of documents through high-level content descriptions. The availability

of high-level content descriptions for documents will enable information retrieval using more abstract terms, which is crucial for providing more valuable services. The media interpretation framework exploits various reasoning services whereas the abductive retrieval inference service offered by RACER plays the key role. The overall goal of the framework is to maximize precision and recall of semantics-based information retrieval [MHN98].

Abduction is usually described as a form of reasoning from effects to causes. Another widely accepted definition of abduction considers it as inference from observations to explanations. In this view, abduction aims to find explanations for observations. In general, abduction is formalized as follows: $\Sigma \cup \Delta \models \Gamma$ where background knowledge (Σ), and observations (Γ) are given and explanations (Δ) are to be computed.

If DLs are used as the underlying knowledge representation formalism [Baa03], Σ is a knowledge base (KB): $\Sigma = (\mathcal{T}, \mathcal{A})$ that consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. Δ and Γ are ABoxes and they contain sets of concept instance and role assertions.

We consider ABox abduction in DLs as the key inference service for media interpretation. We assume $\mathcal{A}$ to be empty and modify the previous equation to $\Sigma \cup \Gamma_1 \cup \Delta \models \Gamma_2$, by splitting the assertions in Γ into two parts: bona fide assertions (Γ_1) and assertions requiring fiats (Γ_2). Bona fide assertions are assumed to be true by default, whereas fiat assertions are aimed to be explained.

In order to compute explanations, ABox abduction can be implemented as a non-standard retrieval inference service in DLs. Different from the standard retrieval inference services, answers to a given query cannot be found by simply exploiting the knowledge base. In fact, the abductive retrieval inference service has the task of acquiring what should be added to the knowledge base in order to positively answer a query.

To answer a given query, the abductive retrieval inference service can exploit non-recursive DL-safe rules with autoepistemic semantics in a backward-chaining way. In this approach, rules are part of the knowledge base and are used to extend the expressivity of DLs. In order to extend expressivity and preserve decidability at the same time, the safety restriction is introduced for rules. Rules are DL-safe if they are only applied to ABox individuals, i.e., individuals explicitly named in the ABox [MN07]. In [PKM+07] we presented a detailed discussion of the abductive retrieval inference service in DLs.

The output of the abductive retrieval inference service should be a set of explanations Δ that are consistent w.r.t. Σ and Γ. This set, which is called Δs, is transformed into a poset according to a preference score. We propose the following formula to compute the preference score of each explanation: $S(\Delta) := S_i(\Delta) - S_h(\Delta)$ where S_i and S_h are defined as follows:

$$S_i(\Delta) := |\{i | i \in \mathit{inds}(\Delta) \text{ and } i \in \mathit{inds}(\Gamma_1)\}|$$
$$S_h(\Delta) := |\{i | i \in \mathit{inds}(\Delta) \text{ and } i \in \mathit{newInds}\}|$$

The set *newInds* contains all individuals that are hypothesized during the generation of an explanation (new individuals). The function *inds* returns the set of all individuals found in a given ABox or a set. The preference score reflects the two criteria proposed by Thagard for selecting explanations [Tha78], namely simplicity and consilience. In fact, the less hypothesized individuals an explanation contains (simplicity) and the more observations an explanation involves (consilience), the higher its preference score gets.

7.1 The Media Interpretation Framework

The media interpretation framework aims to compute high-level content descriptions of media documents from lower level information extraction results. For this purpose, it exploits conceptual and contextual knowledge (see Figure 25). Here, the contextual knowledge refers to specific prior knowledge relevant for the high-level interpretation, which we will discuss later. The conceptual knowledge is represented in a formal ontology that consists of a TBox and a set of non-recursive DL-safe rules about the domain of interest. The formal representation of the conceptual knowledge enables the framework to compute interpretations using various reasoning services such as the abductive retrieval inference service presented below.

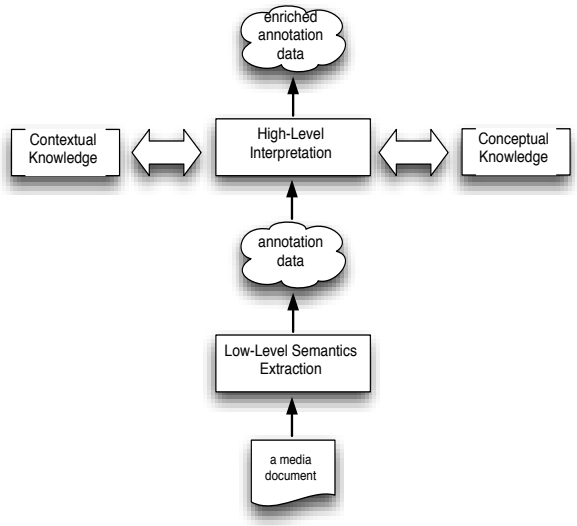


Figure 25: Architecture of the media interpretation framework

The high-level interpretation of a media document requires an ABox as input (analysis ABox), which contains the results of the low-level semantics extraction. It produces another ABox as output (interpretation ABox), which contains high-level content descriptions. The analysis ABox corresponds to Γ in the abduction formula (see Section 7). The interpretation ABox is computed in an iterative process, and at the end of this process it contains all possible interpretations of the media document. Each iteration of the interpretation process consists of the following steps:

First, Γ is split into bona fide and fiat assertions. Currently, all role assertions in the analysis ABox are selected as fiat assertions (Γ_2), and all other assertions as bona fide ones (Γ_1). Second, each assertion from Γ_2 is transformed into a corresponding query to exploit the abductive retrieval inference service. Consequently, the abductive retrieval inference service returns all possible consistent explanations. Third, for each explanation it is checked whether new information can be inferred through deduction.

The interpretation process selects new assertions as fiat assertions from each generated explanation, and repeats these steps until no new explanation can be generated.

Additionally, contextual knowledge can be used to enhance the results obtained by the interpretation process: A set of aggregate concepts can be defined as target concepts.

Target concepts serve as an additional termination criteria to omit the computation of interpretations which are useless in practice. Consequently, the framework terminates the cyclic interpretation process, once a generated explanation contains an instance of the target concepts.

In the future the contextual knowledge can be extended. E.g., more appropriate (probably domain-specific) strategies for identifying fiat assertions can be developed and integrated into the framework.

After the presentation of the media interpretation framework, we discuss the details of the underlying interpretation process using an image and the athletics ontology AEO (see Section 2). The athletics ontology that serves as the background knowledge Σ consists of a TBox and a set of non-recursive DL-safe rules. Some axioms of the TBox, which are relevant for our example are shown below:

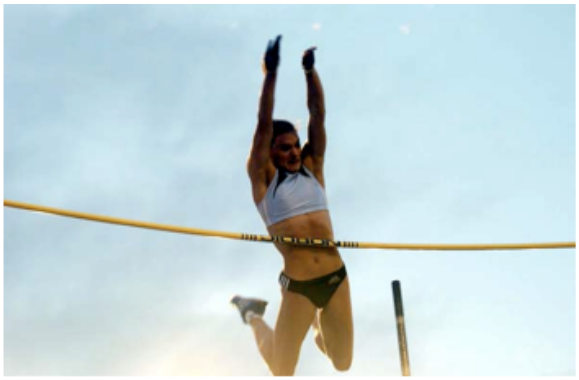
$$
\begin{array}{lcl}
Person & \sqsubseteq & \exists hasPart.PersonFace \sqcap \\
 & & \exists hasPart.PersonBody \sqcap \\
 & & \neg PersonFace \sqcap \ldots \\
Jumper & \sqsubseteq & Person \\
SportsTrial & \sqsubseteq & \exists hasPerformance. \\
 & & Performance \sqcap \\
 & & \exists hasRanking.Ranking \sqcap \\
 & & \exists hasParticipant.Person \\
 & & \neg Person \sqcap \ldots \\
JumpingEvent & \sqsubseteq & SportsTrial \sqcap \\
 & & \exists_{\leq 1} hasParticipant.Jumper \\
PoleVault & \sqsubseteq & JumpingEvent \sqcap \\
 & & \exists hasPart.Pole \sqcap \\
 & & \exists hasPart.Bar \\
HighJump & \sqsubseteq & JumpingEvent \sqcap \\
 & & \exists hasPart.Bar
\end{array}
$$

In this TBox, some concepts such as *Person* are more abstract than others, and are designed as aggregates, which consist of parts such as *PersonFace* and *PersonBody*. Furthermore, the TBox contains several disjointness axioms between concepts, which are not shown here completely for brevity. The disjointness axioms are necessary to avoid 'awkward' explanations, which would otherwise be generated.

Additionally, the background knowledge contains a set of non-recursive DL-safe rules that are used to model several characteristic constellations (relations) of objects in the athletics domain as follows:

$$
\begin{array}{lcl}
adjacent(Y,Z) & \leftarrow & Person(X), hasPart(X,Y), \\
 & & PersonFace(Y), hasPart(X,Z), \\
 & & PersonBody(Z) \\
adjacent(Y,Z) & \leftarrow & PoleVault(X), hasPart(X,Y), \\
 & & Bar(Y), hasPart(X,W), \\
 & & Pole(W), hasParticipant(X,Z), \\
 & & Jumper(Z) \\
adjacent(Y,Z) & \leftarrow & HighJump(X), hasPart(X,Y), \\
 & & Bar(Y), hasParticipant(X,Z), \\
 & & Jumper(Z) \\
adjacent(X,Z) & \leftarrow & hasPart(X,Y), adjacent(Y,Z)
\end{array}
$$

To better illustrate the interpretation process and the use of the background knowledge, we continue with the stepwise interpretation of an athletics image. The image below shows a pole vault trial:



Yelena Isinbayeva goes over 5.01m but knocks off the bar on her descent (Hasse Sjögren)

Assume that for this image low-level image analysis delivers an analysis ABox with the following concept instance and role assertions:

$\Gamma = \{pface_1 : PersonFace,\ pole_1 : Pole,\ bar_1 : Bar,\ pbody_1 : PersonBody,\ (pface_1,\ pbody_1) : adjacent,\ (pbody_1,\ bar_1) : adjacent\}$

To begin with the interpretation, all role assertions are selected as fiat assertions and, therefore, Γ_2 becomes:

$\Gamma_2 = \{(pbody_1,\ bar_1) : adjacent,\ (pface_1,\ pbody_1) : adjacent\}$

In the second step, the role assertions are transformed into corresponding queries and the abductive retrieval inference service is asked for explanations. Only the query derived from the role assertion $(pface_1, pbody_1) : adjacent$ results in the generation of an explanation. It explains the adjacency of the face and the body by hypothesizing a person instance to whom they both belong to (see the first *adjacent* rule). Note that other *adjacent* rules are considered as well, however they cause the generation of explanations that are inconsistent (due to the disjointness axioms in the TBox). The interpretation process discards such explanations. Assume that the newly inferred person instance is named new_ind_1. In the third step, the interpretation process applies the rules forwards to check whether new information can be deduced. This yields the following assertions: $(bar_1,\ new\_ind_1) : adjacent$, $(pbody_1,\ new\_ind_1) : adjacent$.[17] At this state, the interpretation process defines a new Γ_2 by selecting all newly inferred role assertions as fiat assertions and repeats the whole cycle. Here, only the query derived from the role $(bar_1,\ new\_ind_1) : adjacent$ results in the generation of explanations:

- $\Delta_1 = \{new\_ind_2 : PoleVault,\ (new\_ind_2,\ bar_1) : hasPart,\ (new\_ind_2,\ pole_1) : hasPart,\ (new\_ind_2,\ new\_ind_1) : hasParticipant,\ new\_ind_1 : Jumper\}$

[17] See the fourth *adjacent* rule

- $\Delta_2 = \{new\_ind_3 : HighJump,\ (new\_ind_3,\ bar_1) : hasPart,$
  $(new\_ind_3,\ new\_ind_1) : hasParticipant,$
  $new\_ind_1 : Jumper\}$

At this point, no further explanations can be generated and the interpretation process terminates. Observe that both explanations are consistent and represent possible interpretations of the image. However, in practice one would like to get 'preferred' explanation(s) only. For this purpose, the preference score presented in Section 7 can be used. The preference score of Δ_1 is calculated as follows: Δ_1 incorporates the individuals bar_1, $pole_1$ and new_ind_1, and therefore $S_i(\Delta_1)$=3. Furthermore, it hypothesizes only one new individual, namely new_ind_2, such that $S_h(\Delta_1)$=1. The preference score of Δ_1 is therefore $S(\Delta_1) = S_i(\Delta_1)$ - $S_h(\Delta_1) = 2$. Analogously, the preference score of the second explanation is $S(\Delta_2)$=1. Consequently, Δ_1 becomes the 'preferred' explanation for the image. In fact, the result is plausible, since this image should better be interpreted as showing a pole vault and not a high jump, due to the fact that image analysis could detect a pole, which should not be ignored as in the high jump explanation (consilience).

7.2 Test Setup

The overall goal of the framework is to provide high-level content descriptions of media documents for maximizing precision and recall of semantics-based information retrieval. In this subsection, we provide an empirical evaluation of the results of the framework on a collection of athletics images in order to analyze the utility of the framework.

For this purpose, we implemented the media interpretation framework shown in Figure 25. The core component of this implementation is the DL-reasoner RACER [HMW07] that supports various inference services. The abductive retrieval inference service, which is the key inference service for media interpretation, is integrated into the latest version of RACER. The framework gets analysis ABoxes, exploits various inference services of RACER, and returns interpretation ABoxes as high-level content descriptions. For the time being, the computation of preference scores is not implemented and, therefore, interpretation ABoxes contain all possible explanations.

To test the implementation, we used an ontology about the athletics domain and an image corpus. The corpus consists of images showing either a pole vault or a high jump event. The images have been manually annotated with annotation tools in order to train low-level feature extractors for prospective athletics corpora. I.e., using the annotation tools, annotators manually annotated regions of images (as visual representations of concepts), with corresponding concepts from the ontology such as *Pole*, *Bar* and *PersonFace*. Afterwards, annotated images have been analyzed automatically to detect relations between concept instances. Finally, for each image in the corpus an analysis ABox with corresponding assertions has been generated.

We tested the implementation in the following setup: the aggregate concepts *PoleVault* and *HighJump* from the domain ontology are defined as target concepts. Analysis ABoxes of pole vault and high jump images are used as input for high-level media interpretation.

7.3 Test Results

The results obtained for pole vault and high jump images are shown in Figure 26 and 27, respectively. To analyze the usefulness of the results for information retrieval, in both figures interpretation ABoxes are categorized w.r.t. the existence (or absence) of aggregate concept instances: A) contains no aggregate concept instances at all B) contains an aggregate concept instance but no target concept instance C) contains a *HighJump* and a *PoleVault* instance D) contains a *PoleVault* instance E) contains more than one *PoleVault* instances and one or no *HighJump* instances

At first sight, only interpretation ABoxes that fall into the category D in Figure 26 look like 'good' interpretation results for pole vault images, because the corresponding images are annotated with a single *PoleVault* instance. However, if the implementation would be enhanced to include preference scores, as discussed in Section 7.1 for an example pole vault image, all interpretation ABoxes of category C and E would include the most 'preferred' explanation only (in this case a single *PoleVault* instance), and hence fall into the category D, too.

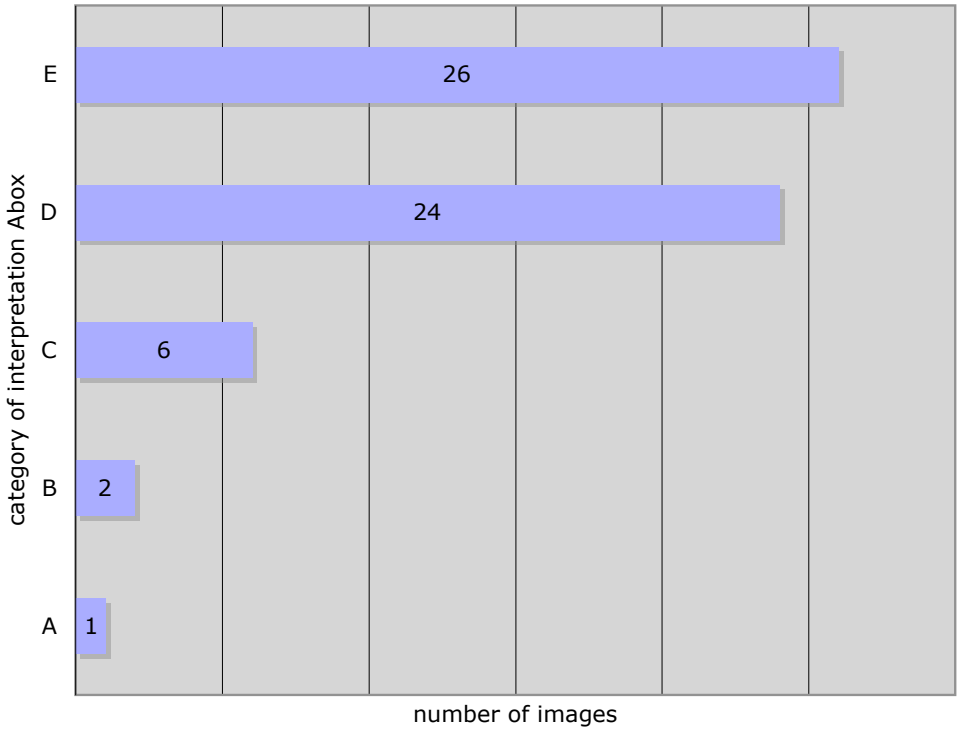


Figure 26: Results for pole vault images.

Both in Figure 26 and 27, category A interpretation ABoxes are identical to the corresponding analysis ABoxes and indicate that no new knowledge could be inferred through high-level interpretation. For other images (category B interpretation ABoxes) high-level interpretation infers new knowledge (including an aggregate concept instance) but fails to derive an instance of the target concepts.

In fact, category B interpretation ABoxes contain a *Person* instance to explain the existence of *PersonBody* and *PersonFace* instances and their constellation in the image.

Deeper analysis of category *A* and *B* interpretation ABoxes showed that insufficient interpretation results are caused by the failure of image analysis to extract some of the existing relations in the corresponding images. Taking into account the ambiguity and uncertainty involved in the image analysis process, this information (the failure of adequate interpretation) can be used to create a valuable feedback for the image analysis tools.

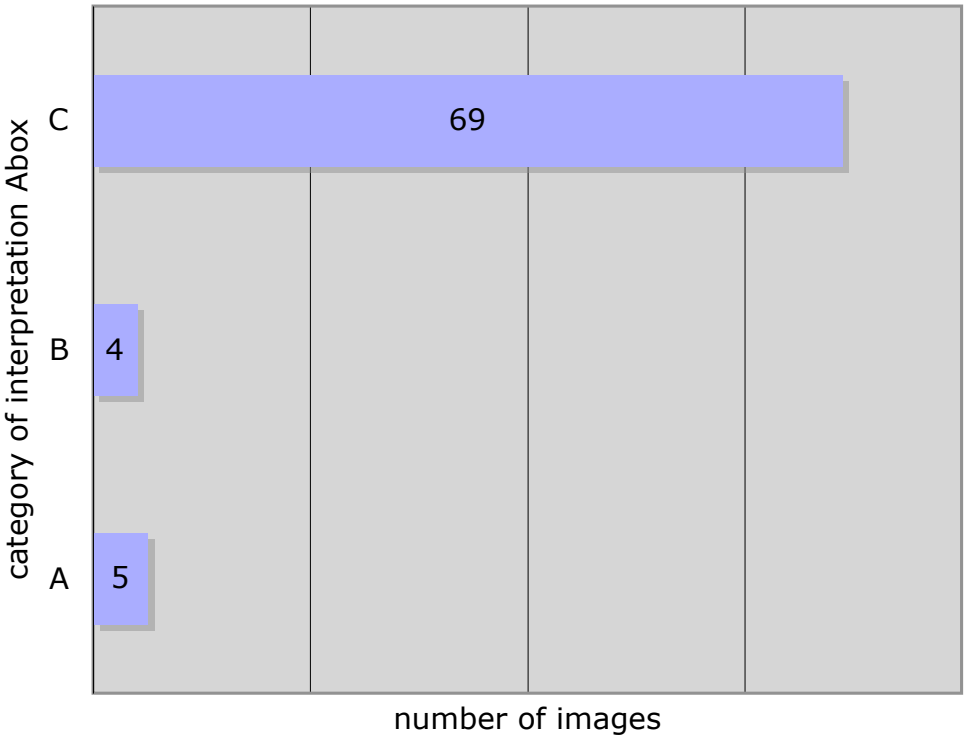


Figure 27: Results for high jump images.

Figure 27 shows that every high jump image is interpreted as either showing a high jump or a pole vault event (category C), besides incompletely analyzed ones, which fall into the categories *A* or *B*. Different from pole vault images, interpretations of high jump images cannot be disambiguated through preference scores. This result indicates that necessary rules are missing in the background knowledge due to the fact that, currently, image analysis cannot extract distinctive features of high jump images.

Our experiments showed that, if provided with an appropriate ontology and low-level annotations, the existing implementation of the media interpretation framework delivers promising results for images and can be used for maximizing precision and recall of semantics-based information retrieval systems.

8 Non-Standard Inferences

The name Sonic stands for "simple ontology non-standard inference component". This system implements a whole collection of non-standard inferences.

8.1 Sonic

In its current version Sonic implements a range of so-called non-standard inferences. Sonic comprises basically two parts. One is the Sonic reasoner, which implements the non-standard inferences. The other part is ontology editor component that realizes a graphical user interface to access the inferences in an easy way. We will concentrate in this deliverable and report on those inferences that are helpful in realizing ontology design and maintenance tasks as described in the TONES deliverable D05.

Generating Concept Descriptions. The ontology designer wants to add a new concept to the ontology, but finds it difficult to describe it. To obtain a starting point for the concept description, the designer wants to automatically generate an initial description of the new concept that is based on the position of this concept in the subsumption hierarchy.

Structuring the Ontology. The ontology designer wants to improve the structure of an ontology by inserting intermediate concepts into the subsumption hierarchy. He needs support to decide where to add such concepts and how to describe them.

Bottom-up Construction. The ontology designer wants to design the ontology bottom-up, i.e., by proceeding from the most specific concepts to the most general ones. This should be supported by automatically generating concept descriptions from descriptions of typical instances of the new concept.

Ontology Customization. An ontology user wants to adapt an existing ontology to her purposes by making simple modifications. Since she is not an expert in ontology languages, she works with a simpler language than the one used to formulate the ontology and/or with graphical frame-like interfaces.

Concept Inspection. The ontology designer wants to display a concept description in a way that facilitates understanding of the concept's meaning.

The inference central to most of these tasks is the computation of common subsumers—either the computation of least common subsumers for the structuring of the ontology and the bottom-up construction or the computation of good common subsumers employed in ontology customization. In case disjunction is present in the DL in use, the least common subsumer is simply the disjunction of the input concepts. The disjunction is not a good starting point for the modeler to edit the concept description, since it does not extract the commonalities from the input concepts, but merely enumerates them. To remedy this two approaches have been proposed on which we concentrated in our testing. More precisely, we tested our implementation of the approximation-based method to obtain "meaningful" common subsumers in the presence of disjunction and two methods for obtaining good common subsumers for the customization of background ontologies.

8.2 Test Setup

For our tests we concentrated on the computation of common subsumers. More precisely, we tested our implementation of the approximation-based method to obtain "meaningful" common subsumers in the presence of disjunction and two methods for obtaining good common subsumers for the customization of background ontologies.

	nr. of concepts	nr. of definitions	nr. of test tuples
DICE	3500	3249	75
OntoCAPE	588	575	60

Table 9: SONIC test data overview.

8.2.1 Test data

We tested SONIC on two ontologies from practical applications:

- The DICE ontology models concepts from the medical domain, more precisely it describes reasons for the admission to intensive care. This ontology was introduced in the TONES deliverable 14 [CGG+07b].
- The OntoCAPE ontology models concepts from the domain of chemical process engineering and was described in Section 2.8.

For both TBoxes we used versions that pruned the expressivity to the concept constructors that SONIC can handle, i.e., role declarations were omitted and number restrictions were removed in the versions of the TBoxes used in our tests.

To evaluate the approaches for the computation of common subsumers in presence of disjunction and the inferences that realize them, we need sets of concepts that we can use as input for the common subsumer inferences in our tests. We selected these input sets by first classifying the test ontology and then identifying concepts with many concept children, i.e., direct subsumees. From this set of direct subsumees we picked subsets randomly, which are the input test data for our evaluation of the common subsumer inferences.

The idea behind this way of selecting the input is to simulate the application of the bottom-up approach, where unbalanced concept hierarchies are augmented with new concepts to obtain a more tree-like concept hierarchy by introducing a new parent concept for sibling concepts. So, by identifying concepts with many concept children, we focus on a part of the concept hierarchy that a knowledge engineer might select for an extension by an intermediate concept. Moreover, by this way of selecting the input sets, we guarantee that no trivial common subsumers are obtained, that collapse to $\top$. We would always obtain at least the common parent concept—thus the computation is not completely trivial.

We picked 75 such concept sets in the above described fashion randomly from the $\mathcal{ALC}$-version of the DICE ontology and 60 such sets from $\mathcal{ALC}$-version of the OntoCAPE ontology. Each of the sets contains 2 to 7 concept names.

We ran our tests on a standard PC with 500MB of memory under Linux. The Lisp source code for the inferences was compiled and ran under ACL 8.0. As the underlying standard reasoner we used RACERPRO (Version 1.9.1) also compiled under ACL 8.0.

8.3 Test Results

8.3.1 Evaluation of the precision of common subsumers

To evaluate the usefulness of the concept descriptions obtained by the inferences for computing common subsumers, we concentrate on the precision of the obtained result. Here, precision is to be understood in terms of information loss between the disjunction of the input concepts—the trivial least common subsumer—and the concept description obtained by applying one of the techniques for the 'meaningful common subsumer'. We proceed by evaluating the precision of the two approaches individually.

To assess the precision of the concept descriptions obtained by the common subsumer we proceed by testing whether the trivial lcs, i.e., the disjunction of the input concepts, is equivalent to the common subsumer obtained by the approximation-based approach, the scs or the acs. If not, we can estimate which information was lost for the approximation-based approach in a second step by computing the difference for the unfolded trivial lcs and the approximation-based lcs, i.e., the concept description obtained by first approximating each input concept description in $\mathcal{ALE}$ and then computing their lcs.

In our setting the acs yields equivalent concept descriptions to the ones obtained by the approximation-based approach. The acs is obtained by unfolding the $\mathcal{ALE}(\mathcal{T})$-input concepts[18] completely, yielding $\mathcal{ALC}$-concept descriptions and then applying the $\mathcal{ALE}$-approximation to the disjunction of the unfolded input concept descriptions. Although the concept descriptions obtained by the two methods need not be the same syntactically, the evaluation of the approximation-based approach carry over to the acs to some extent.

For the subsumption closure-based common subsumers (scs) computed w.r.t. a background terminology we cannot use the difference operator to asses the information loss, in case the common subsumer obtained by these methods is more general than the trivial lcs. To see the reason for this, consider the following example where the TBox is $\mathcal{T} = \{A = B \sqcup C\}$ and we are interested in the lcs of $C_1 = B \sqcap \exists r.D$ and $C_2 = C \sqcap \exists r.E$. Then the $\mathsf{lcs}_{\mathcal{ALC}}(C_1, C_2) = C_1 \sqcup C_2$, while the $\mathsf{scs}_{\mathcal{ALE}(\mathcal{T})}(C_1, C_2) = A \sqcap \exists r.D \sqcap \exists r.E$. Both concept descriptions are equivalent, but the difference operator would return the syntactic difference, between them. The syntactic difference is in this case misleading to assess the information loss. Applying the difference operator to the unfolded concept description obtained by the scs is not possible either, since it is an $\mathcal{ALC}$-concept descriptions for which we would need a difference operator that can compute the syntactic difference between $\mathcal{ALC}$-concept descriptions.

Precision of the approximation-based approach To evaluate the precision of the approximation-based approach, we computed for each concept set $S = \{C_1, \ldots, C_n\}$ in the test data the following concept descriptions:

1. the trivial lcs: the disjunction of the concepts in the concept set unfolded w.r.t. the underlying TBox: $\mathsf{lcs}_{\mathcal{ALC}}(C_1, \ldots, C_n) = \mathsf{unfold}(\bigsqcup_{1 \leq i \leq n} C_i)$.

[18] Recall that the concept descriptions use concept concept constructors from $\mathcal{ALE}$, but may used concept names from the background TBox.

	$\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{lcs}_{\mathsf{approx}}$
DICE	36 (48,0%)
OntoCAPE	8 (12,3%)

Table 10: Comparison of $\mathsf{lcs}_{\mathcal{ALC}}$ and $\mathsf{lcs}_{\mathsf{approx}}$.

	$\lvert\mathsf{lcs}_{\mathcal{ALC}}\rvert$	$\lvert\mathsf{lcs}_{\mathsf{approx}}\rvert$	$\lvert\mathsf{diff}(\mathsf{lcs}_{\mathcal{ALC}}, \mathsf{lcs}_{\mathsf{approx}})\rvert$	$\mathsf{diff}(\dots) \equiv \top$
DICE	68,1	7,4	14,7	39 (52,0%)
OntoCAPE	32,2	2,2	15,4	50 (83,3%)

Table 11: Applying the difference operator to $\mathsf{lcs}_{\mathcal{ALC}}$ and $\mathsf{lcs}_{\mathsf{approx}}$.

2. the approximation-based lcs: the $\mathcal{ALE}$-lcs of the set of $\mathcal{ALE}$-approximations of each $\mathcal{ALC}$-concept from the concept set:
   $\mathsf{lcs}_{\mathsf{approx}}(C_1, \dots, C_n) = \mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{approx}_{\mathcal{ALE}}(C_i) \mid 1 \leq i \leq n\}\big)$.

3. the syntactic difference of the trivial lcs and the approximation-based lcs:
   $D_{\mathsf{approx}} = \mathsf{diff}(\mathsf{lcs}_{\mathcal{ALC}}(C_1, \dots, C_n),\ \mathsf{lcs}_{\mathsf{approx}}(C_1, \dots, C_n))$.

We ran these tests for the $\mathcal{ALC}$-variants of the DICE and the OntoCAPE test ontology. Table 10 shows in the first column the number of cases where the trivial lcs is strictly more specific than the concept description obtained by the approximation-based approach. In these cases information captured in the trivial lcs, common to all input concept descriptions was lost when computing the lcs of the $\mathcal{ALE}$-aproximations of the concept descriptions. It shows that information is lost in 48% of the cases tested for the DICE TBox and 12,3% for the OntoCAPE TBox.

The Table 11 shows the average concept size of the trivial lcs of the approximation-based lcs and of their difference obtained by the heuristic for computing the difference. It shows for both test TBoxes that the difference between the $\mathsf{lcs}_{\mathcal{ALC}}$ and the $\mathsf{lcs}_{\mathsf{approx}}$ results in concept descriptions a couple of times larger that the $\mathsf{lcs}_{\mathsf{approx}}$ itself. This might seem a daunting result at first, but recall that the heuristic for computing the difference applied to concept description with redundancy yields a syntactic difference with redundancies. In fact, we obtained a difference equivalent to $\top$ in the vast majority of the cases for both TBoxes (see last column). Thus the concept sizes for the difference give a biased picture of the quality of $\mathsf{lcs}_{\mathsf{approx}}$.

In 11 of the test cases for the DICE TBox we obtained a concept name as the result of applying the approximation-based approach, which indicates that the common parent concept of the concepts from the tuple was obtained as their common subsumer. For the OntoCAPE knowledge base 11 such cases were found. in regard of our application scenario these are the cases where no new node is introduced in the concept hierarchy and the modeler would have to revise her choice of input concepts.

Precision of the common subsumers for background ontologies For the evaluation of the precision of the common subsumers computed for the customization of background terminologies we examine the same quality criteria as above for the computation

	$\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{scs}$	$\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{acs}$	$\mathsf{scs} \sqsubset \mathsf{acs}$
DICE	-	36 (48,0%)	36 (48,0%)
OntoCAPE	2 (3,1%)	8 (12,3%)	52 (80,0%)

Table 12: Subsumption relationships between $\mathsf{lcs}_{\mathcal{ALC}}$, acs and scs.

of the scs and the acs.

To asses the precision in this setting, we can only refer to the subsumption relationships between the obtained concept descriptions, since the difference operator does not yield meaningful results in this setting for the reasons explained earlier. We computed for each concept set $S = \{C_1, \ldots, C_n\}$ in the test data:

1. the trivial lcs: the disjunction of the concepts in the concept set unfolded w.r.t. the underlying TBox: $\mathsf{lcs}_{\mathcal{ALC}}(C_1, \ldots, C_n) = \mathsf{unfold}(\bigsqcup_{1 \leq i \leq n} C_i)$.

2. the approximation-based gcs: the acs of the disjunction of $\mathcal{ALE}(\mathcal{T})$-concepts from the concept set:
   $\mathsf{acs}(C_1, \ldots, C_n) = \mathsf{approx}_{\mathcal{ALE}}(\bigsqcup_{1 \leq i \leq n} C_i)$

3. the subsumption closure-based gcs: the scs of the concept set:
   $\mathsf{scs}(C_1, \ldots, C_n)$

We computed these concept descriptions for the tuples from the DICE and the OntoCAPE test data. We checked for the subsumption relations between the obtained concept descriptions. The results are displayed in Table 12. The first two columns show the number of cases, where scs (acs) is more general then the trivial lcs. It shows, that the scs is only in two cases more general than the lcs and thus does result in hardly any information loss for our test data.

For the acs, we obtain the same information loss, as for the approximation-based approach. Interestingly, the scs results always in a more specific concept description than the acs, if the two are not equivalent. This is somewhat different from the results in [BST07], where also cases appeared in which the acs was more specific than the scs.

In this setting we obtained only 8 trivial acs concept descriptions, i.e., concepts that collapsed to the common parent concept. For the scs this number of collapsed concepts is 2. Regarding the precision of common subsumers, the scs showed the best performance on our test data.

8.3.2 Performance of the computation of common subsumers

In Table 13 we see the average run-times measured for the different ways to obtain common subsumers. These run-times were obtained by using an implementation that realizes lazy unfolding. It shows that with this optimization technique applied alone the run-times for our examples from practical applications are in most cases below 1,5 seconds. This is already an acceptable run-time for interactive use—where run-time measured for $\mathsf{lcs}_{\mathsf{approx}}$ might be seen as an exception. However, it is, again the scs that shows the

best performance, when comparing the three approaches to obtain non-trivial common subsumers. It only uses a fraction of the run-times of the other common subsumers. Surprisingly, the computation of the scs is even faster than the trivial lcs. This effect is due to $\mathcal{ALE}$-unfolding, where the input concept description is not necessarily unfolded completely (if concept definitions are encountered that cannot be transformed into an $\mathcal{ALE}$-concept by De Morgan' rules), while the trivial lcs is obtained by unfolding the disjunctions of the input concepts completely.

The results also indicate that concept approximation is the inference that would benefit most from an optimized implementation. In order to be able to apply conjunct-wise computation for approximation, we have to test whether a concept description is nice– according to conditions specified in [TB07]. Although the conditions for this test have been relaxed, the question still is: do the concept definitions from application knowledge bases contain nice concepts? An investigation of the DICE and the OntoCAPE knowledge base showed that nice concepts do appear in knowledge bases from applications [TB07]. In case of the DICE knowledge base, 13,2% of the concepts are nice. The OntoCAPE knowledge base contains even about 35% of nice concepts. Thus conjunct-wise approximation might help to obtain better run-times for approximation computed w.r.t. knowledge bases obtained from practical applications.

Our evaluation of the common subsumer approaches showed that the common subsumers obtained by applying the lcs to the concepts obtained by approximation performs well. In more than half of the cases the approximation-based approach captures the full information of the trivial lcs. Similarly, the here proposed approaches for the computation of common subsumers w.r.t. a background knowledge base performed well w.r.t. precision of the result. While the acs yielded concept descriptions that capture the information common to all input concepts completely in at least more than the half of the cases, the scs turned out to miss hardly any information on our test cases. Moreover, comparing the two approaches for obtaining good common subsumers w.r.t. a background knowledge base, it showed that the scs yields a more specific concept description than the acs in up to 80% of the cases.

All of the three implementations showed run-times suitable for interactive use, despite their high computational complexity. The scs showed also the best performance for computation times of the three common subsumers. To sum up, the scs seems to be an excellent alternative for the $\mathcal{ALE}(\mathcal{T})$-lcs for which we could not devise a constructive computation method so far.

	$\mathsf{lcs}_{\mathcal{ALC}}$	$\mathsf{lcs}_{\mathsf{approx}}$	acs	scs
DICE	1,34	6,52	1,39	0,23
OntoCAPE	0,15	0,29	0,15	0,03

Table 13: Average run-times of $\mathsf{lcs}_{\mathcal{ALC}}$, $\mathsf{lcs}_{\mathsf{approx}}$, acs and scs (in s).

9 Knowledge Base Completion

9.1 InstExp

InstExp (INSTance EXPlorer) is a DL knowledge base completion tool developed as an extension to version v2.3 beta 3 of the Swoop ontology editor [KPS+06]. It is implemented in the Java programming language, and it communicates with the reasoner over the OWL API [BVL03]. InstExp is available under `http://lat.inf.tu-dresden.de/~sertkaya/InstExp/`. The development of InstExp was partially supported by the EU projects TONES (IST-2005-7603 FET) and Semantic Mining (NoE 507505), and the German Research Foundation DFG (GRK 334/3).

InstExp aims to support enriching an ontology by asking questions to a domain expert. It asks questions of the form "*Is it true that objects that instances of the classes A,B and C also instances of D and E?*". The domain expert is then expected to answer "yes" or "no". If she answers with "no", then she is expected to provide a counterexample, and this counterexample is added to the ontology. If she answers "yes", then the ontology is updated with a new inclusion axiom. When the process stops, the ontology is complete in a certain sense. InstExp implements an extension of the well-known knowledge acquisition method of Formal Concept Analysis, namely *attribute exploration*. The advantage of this method is that it guarantees to ask the minimum number of questions to the expert in order to acquire the missing part of the knowledge. The theoretical background of InstExp was explained in detail in [BGSS06, BGSS07].

A DL knowledge base completion process using InstExp can briefly be sketched as follows: After loading a knowledge base into Swoop and classifying it, the user can start InstExp from the Swoop menu. At this point InstExp displays the concept hierarchy of the loaded knowledge base, and waits the user to select the "interesting" concepts that should be involved in the completion process. As soon as the user finishes selecting these concepts, InstExp displays the individuals in the ABox that are instances of these concepts, and the completion process starts with the first question. The user can confirm or reject the questions by clicking the relevant buttons. If she rejects a question, InstExp displays a counterexample editor, which contains the "potential counterexamples" to the current question, i.e., individuals in the ABox that can be modified to act as a counterexample to this question. The user can either modify one of these existing individuals and turn it into a counterexample, or introduce a new individual into the ABox. During counterexample preparation, InstExp tries to guide the user as follows: If she makes the description of an individual inconsistent, InstExp gives a warning and does not allow her to provide this as the description of a counterexample. Once she has produced a description that is sufficient to act as a counterexample, InstExp notifies the user about it, and allows this description to be added to the ABox.

9.2 Test Setup

The aims of our tests were to evaluate performance and usability of InstExp . More precisely, in our tests we aimed for evaluating performance of InstExp both in terms of runtime and memory usage, and also usability of InstExp in general.

In D14, it was mentioned that InstExp has its rôle in the ontology completion usage scenario. As input, this usage scenario expects a well established ontology written in a DL that supports conjunction and negation, and has a TBox formalism that allows for GCIs. Also, an expert from the application domain is required to answer the questions asked. The expected output for the scenario is the input ontology enriched with new subsumption relationships and new instances that are acquired from the expert as answer to the questions asked. The ontology given as input is expected to have an ABox, and usually a large number of individuals in the ABox. For this reason from the ontologies mentioned in D14, we evaluted our tool on the ontologies that are mentioned to have ABox, namely *Semintec* ontology about financial services, and an ontology generated by the Data Generator(UBA) of Lehigh University Benchmark. Using the generator, we generated an ontology with 14 concept definitions, and 1555 individuals in the DL $\mathcal{AL}(\mathcal{D})$. We also made tests on two smaller fragments of the Semintec ontology that were provided on the Semintec project web page. The first fragment[19] is the part of the Semintec financial ontology, containing information only about gold credit card holders. It contains 39 concept definitions and 297 individuals. The expressivity of the DL used is shown to be $\mathcal{ALCIF}$ by Pellet. The second one[20] obtained from the first fragment by removing disjunctions contains also 39 concept defintions and 297 individuals.

We performed the tests on a computer with 1.4MHz Intel(R) Pentium(R) M processor and 512MB of main memory, running a GNU/Linux operating system with 2.6.18-4 kernel.

9.3 Test Results

In our tests, we have observed that the performance of InstExp heavily depends on the knowledge base to be completed, and the efficiency of the DL reasoner used. As already mentioned, whenever a question is accepted, a new GCI is added to the TBox. This requires the knowledge base to be reclassified. Depending on the size and complexity of the knowledge base, and efficiency of the underlying DL reasoner, this can take long time for knowledge bases of big sizes, which means that the user may have to wait several minutes between two consecutive questions. To some extent this problem can be overcome by using a DL reasoner that can do incremental reasoning, i.e., that can efficiently handle the added GCI and reclassify the knowledge base without starting from the scratch. Pellet can do incremental reasoning to some extent.

9.3.1 Results on the Semintec ontology

Our tests showed that classification of the Semintec ontology takes around 3 minutes using the Pellet reasoner. Since our completion tool is using Pellet as the underlying reasoner, the waiting time between two consequetive questions for this ontology were around 3 minutes on average. Upon starting exploration, the memory usage was 402MB, and after answering 5 questions, it was around 470MB. However, we were not able to measure precisely how much of this memory is required by InstExp , since it is built into Swoop, and is not running as a seperate process.

[19] available under `http://www.cs.put.poznan.pl/alawrynowicz/goldDLP.owl`
[20] available under `http://www.cs.put.poznan.pl/alawrynowicz/goldDLP2.owl`

In order to avoid the long waiting times between questions, we also evaluated InstExp on the two smaller fragments of the Semintec ontology mentioned above. The first fragment, which contains 39 concept definitions and 297 individuals, was classified in 1-2 seconds by Pellet. As a result, the waiting times between two consequetive questions asked by InstExp were also around 1-2 seconds on the average. When we started InstExp , the memory usage was around 56MB. After answering around 10 questions, the memory usage was 64MB. Due to the reason mentioned above, we were not able to measure how much of this memory is used by InstExp , and how much is used by Swoop.

The second fragment is obtained from the first one by removing disjunctions. As the first one, it also contains 39 concept definitons, 297 individuals, and 1 GCI axiom. The test results on this ontology were similar to the first fragment of Semintec in the previous paragraph. Classifying it with Pellet took around 1 second, thus waiting time between two questions asked by InstExp was also around 1 second. Upon starting InstExp , memory usage was around 58MB, and after 10 questions, it was around 65MB.

9.3.2 Results on the UBA-generated ontology

This ontology was classified by Pellet in in 2 seconds, thus waiting time between two questions asked by InstExp was also around 2 seconds. The memory usage upon starting InstExp was 76MB, after answering 10 questions, it was around 84MB. Due to the reason mentioned above, we were not able to measure precisely whether the increase was due to Swoop, or due to InstExp .

9.3.3 Usability of InstExp

One important point we have observed is that during completion, unsurprisingly the expert sometimes makes errors when answering the questions. In the simplest case, the error makes the knowledge base inconsistent, which can easily be detected by DL reasoning and the expert can be notified about it. However, in this case an explanation for the reason of inconsistency is often needed to understand and fix the error. The situation gets more complicated if the error does not immediately lead to inconsistency, but the expert realizes in the later steps that she has done something wrong in one of the previous steps. In this case the tool should be able to help the expert to detect which one of the previous answers leads to the error. Once the source of the error is found, the next task is to correct it without producing extra work for the expert. More precisely, the naive idea of going back to the step where the error was made, and forgetting the answers given after this step will result in asking some of the questions again. The tool should avoid this. More sophisticated approach to minimize the effort for fixing the error cannot be achieved by ad hoc methods, it requires completion algorithm to be modified accordingly.

We have also observed that, in some cases the expert might want to skip a question, and proceed with another one. On the Formal Concept Analysis theory side, this is not an easy task. It needs the particular lexical order used in the algorithm to be modified. Doing it in a naive way might result in loss of soundness or completeness of the algorithm.

10 Conclusion

We have evaluated the tools developed within Workpackages 3 and 4. Our tests of the standard reasoning techniques show that reasoning about ontologies as proposed within the TONES project scales rather well. This holds true in the case of very expressive ontology languages and, to an even larger degree, also for lightweight ontology languages. Our evaluation of the novel reasoning services shows that the computational complexity of the underlying reasoning problems does not prohibit their use on realistic ontologies from practical applications. Although in-depth case studies are out of the scope of this deliveable, which concentrates on testing efficiency, our experiments also suggest that the novel reasoning services provide very useful information and assistance to ontology designers. In several cases, they point out directions for future reasearch that may lead to an even better usability of these services.

References

[Baa03] F. Baader. Description logic terminology. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 485–495. Cambridge University Press, 2003.

[BB93] A. Borgida and R. Brachman. Loading Data into Description Reasoners. *ACM SIGMOD Record*, 22(2):217–226, 1993.

[BBC+07] F. Baader, R. Bernardi, D. Calvanese, A. Calì, B. C. Grau, M. Garcia, G. de Giacomo, A. Kaplunova, O. Kutz, D. Lembo, M. Lenzerini, L. Lubyte, C. Lutz, M. Milicic, R. Möller, B. Parsia, R. Rosati, U. Sattler, B. Sertkaya, S. Tessaris, C. Thorne, and A.-Y. Turhan. Techniques for ontology design and maintenance. Project Deliverable TONES-D13, TONES Consortium, 2007. Available at `http://www.tonesproject.org/`.

[BCG+06] F. Baader, D. Calvanese, G. D. Giacomo, P. Fillottrani, E. Franconi, B. C. Grau, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, R. Möller, B. Parsia, P. Patel-Schneider, R. Rosati, B. Suntisrivaraporn, and S. Tessaris. Formalisms for representing Ontologies: State of the art survey. Project Deliverable TONES-D06, TONES Consortium, 2006. Available at `http://www.tonesproject.org/`.

[BGSS06] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. LTCS-Report LTCS-06-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2006. See http://lat.inf.tu-dresden.de/research/reports.html.

[BGSS07] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In M. M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 230–235. AAAI Press, 2007.

[BHT05] S. Bechhofer, I. Horrocks, and D. Turi. The OWL Instance Store: System Description. In *Proc. of the 20th Int. Conf. on Automated Deduction (CADE-20)*, Lecture Notes in Artificial Intelligence, pages 177–181. Springer, 2005.

[Bre95] P. Bresciani. Querying Databases from Description Logics. In *Proceedings of Knowledge Representation Meets Databases (KRDB'95), Saarbrücken, Germany, DFKI-Research-Report D-95-12*, pages 1–4, 1995.

[BSNS+06] C. Baker, A. Shaban-Nejad, X. Su, V. Haarslev, and G. Butler. Semantic Web Infrastructure for Fungal Enzyme Biotechnologists. *Journal of Web Semantics*, 4(3):168–180, 2006.

[BST07] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logics*, 2007.

[BVL03] S. Bechhofer, R. Volz, and P. W. Lord. Cooking the semantic web with the owl api. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference, (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer, 2003.

[CDGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 2007. To appear.

[CGF+07] D. Calvanese, B. C. Grau, E. Franconi, I. Horrocks, A. Kaplunova, C. Lutz, R. Möller, B. Sertkaya, B. Suntisrivaraporn, S. Tessaris, and A.-Y. Turhan. Software tools for ontology design and maintenance. Project Deliverable TONES-D15, TONES Consortium, 2007. Available at `http://www.tonesproject.org/`.

[CGG+06] D. Calvanese, B. C. Grau, G. D. Giacomo, E. Franconi, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, D. Martinenghi, R. Möller, R. Rosati, S. Tessaris, and A. Turhan. Common framework for representing ontologies. Project Deliverable TONES-D08, TONES Consortium, 2006. Available at `http://www.tonesproject.org/`.

[CGG+07a] D. Calvanese, G. D. Giacomo, B. Glimm, B. C. Grau, V. Haarslev, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, M. Milicic, R. Möller, R. Rosati, U. Sattler, and M. Wessel. Techniques for Ontology Access, Processing, and Usage. Project Deliverable TONES-D18, TONES Consortium, 2007. Available at `http://www.tonesproject.org/`.

[CGG+07b] D. Calvanese, G. D. Giacomo, B. C. Grau, A. Kaplunova, D. Lembo, M. Lenzerini, R. Möller, R. Rosati, U. Sattler, B. Sertkaya, B. Suntisrivaraporn, S. Tessaris, A. Turhan, and S. Wandelt. Ontology-based services: Usage scenarios and test ontologies. Project Deliverable TONES-D14, TONES Consortium, 2007. Available at `http://www.tonesproject.org/`.

[CM77] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the Nineth ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[DFK+07] J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. Scalable Semantic Retrieval Through Summarization and Refinement. In *21st Conference on Artificial Intelligence (AAAI)*, pages 299–304, 2007.

[EHK+07] S. Espinosa, V. Haarslev, A. Kaplunova, A. Kaya, S. Melzer, R. Möller, and M. Wessel. Reasoning Engine Version 1 and State of the Art in Reasoning Techniques. Technical report, Hamburg University Of Technology, 2007. BOEMIE Project Deliverable D4.2.

[EKM+07] S. Espinosa, A. Kaya, S. Melzer, R. Möller, T. Näth, and M. Wessel. Reasoning Engine Version 2. Technical report, Hamburg University Of Technology, 2007. BOEMIE Project Deliverable D4.5.

[FKM+06] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The Summary Abox: Cutting Ontologies Down to Size. In *Proc. of International Semantic Web Conference (ISWC)*, pages 343–356, 2006.

[GH06] Y. Guo and J. Heflin. A Scalable Approach for Partitioning OWL Knowledge Bases. In *Proc. of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006), Athens, Georgia, USA*, pages 47–60, 2006.

[GHP03] Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL repositories. In *Proc. of the Second Int. Semantic Web Conf. (ISWC 2003)*, number 2870 in LNCS, pages 613–627. Springer Verlag, 2003.

[GMUW02] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.

[GPH04] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proc. of the Third Int. Semantic Web Conf. (ISWC 2004)*, volume 3298 of *LNCS*, pages 274–288. Springer Verlag, 2004.

[GPH05] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[HM99] V. Haarslev and R. Möller. An Empirical Evaluation of Optimization Strategies for ABox Reasoning in Expressive Description Logics. In *Proc. of DL99, International Workshop on Description Logics, Linköping*, pages 115–119, 1999.

[HM01a] V. Haarslev and R. Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.

[HM01b] V. Haarslev and R. Möller. The Description Logic ALCNHR+ Extended with Concrete Domains: A Practically Motivated Approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 29–44. Springer-Verlag, 2001.

[HMW07] V. Haarslev, R. Möller, and M. Wessel. RacerPro User's Guide and Reference Manual Version 1.9.1, May 2007.

[KPS+06] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. A. Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 4(2):144–153, 2006.

[MHN98] R. Möller, V. Haarslev, and B. Neumann. Semantics-Based Information Retrieval. In *Proc. IT&KNOWS-98: International Conference on Information Technology and Knowledge Systems, 31. August- 4. September, Vienna, Budapest*, pages 49–6, 1998.

[MHW06] R. Möller, V. Haarslev, and M. Wessel. On the Scalability of Description Logic Instance Retrieval. In C. Freksa and M. Kohlhase, editors, *29. Deutsche Jahrestagung für Künstliche Intelligenz*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2006.

[MN07] R. Möller and B. Neumann. Ontology-based reasoning techniques for multimedia interpretation and retrieval. In *Semantic Multimedia and Ontologies: Theory and Applications*. 2007. To appear.

[MS06] B. Motik and U. Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), Phnom Penh, Cambodia, November 13-17*, volume 4246 of *LNCS*, pages 227–241. Springer, 2006.

[MYM07] J. Morbach, A. Yang, and W. Marquardt. OntoCAPE—A large-scale ontology for chemical process engineering. *Engineering Applications of Artificial Intelligence*, 20(2):147–161, 2007.

[MYQ+06] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards A Complete OWL Ontology Benchmark. In *Proc. of 3rd European Semantic Web Conference (ESWC)*, pages 124–139, 2006.

[PKM+07] S. E. Peraldi, A. Kaya, S. Melzer, R. Möller, and M. Wessel. Multimedia Interpretation as Abduction. In *Proc. DL-2007: International Workshop on Description Logics*, 2007.

[RZH+07] J. Rilling, Y. Zhang, V. Haarslev, W. Meng, and R. Witte. A Unified Ontology-Based Process Model for Software Maintenance and Comprehension. In *Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006), T. Kühne (Ed.), LNCS 4364, Springer-Verlag*, pages 56–65, 2007.

[SNBHB05] A. Shaban-Nejad, C. Baker, V. Haarslev, and G. Butler. The FungalWeb Ontology: Semantic Web Challenges in Bioinformatics and Genomics. In *Semantic Web Challenge - Proceedings of the 4th International Semantic Web Conference, Nov. 6-10, Galway, Ireland, Springer-Verlag, LNCS, Vol. 3729*, pages 1063–1066, 2005.

[TB07] A.-Y. Turhan and Y. Bong. Speeding up approximation with nicer concepts. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, and A.-Y. Turhan, editors, *Proc. of DL 2007*, 2007.

[TBK+06] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Moeller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithoener. DIG 2.0 – Towards a Flexible Interface for Description Logic Reasoners. In B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *OWL: Experiences and Directions 2006*, 2006.

[Tha78] R. P. Thagard. The best explanation: Criteria for theory choice. *The Journal of Philosophy*, 1978.

[WLL+07] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. v. Henke, and O. Noppens. Real-world Reasoning with OWL. In *Proc. European Semantic Web Conference*, 2007.

[WLLB06] T. Weithöner, T. Liebig, M. Luther, and S. Böhm. What's Wrong with OWL Benchmarks? In *Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006), Athens, GA, USA*, 2006.

[WM05] M. Wessel and R. Möller. A High Performance Semantic Web Query Answering Engine. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, pages 84–95. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.

[ZRH06] Y. Zhang, J. Rilling, and V. Haarslev. An Ontology Based Approach to Software Comprehension - Reasoning about Security Concerns in Source Code. In *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), IEEE Computer Society Press*, pages 333–342, 2006.