D4.2: Reasoning Engine Version 1 and State of the Art in Reasoning Techniques

Version 1.0

S. Espinosa[1], V. Haarslev[2], A. Kaplunova[1], A. Kaya[1],
S. Melzer[1], R. Möller[1], M. Wessel[1]

[1]Hamburg University of Technology (TUHH)
[2]Concordia University, Montreal

Project ref.no.:	*FP6-027538*
Workpackage:	*WP4*
WP / Task responsible:	*TUHH*
Other contributors:	
Quality Assurance:	Alfio Ferrara, George Paliouras, Vassilis Tzouvaras
Document type:	*Report*
Security (distribution level):	*Public*
Status & version:	
Contractual date of delivery:	*M10*
Actual date of delivery:	*16-Februrary-07*
Deliverable number:	*D4.2*
Deliverable name:	Reasoning engine version 1 and State of the Art in Reasoning Techniques
Date:	*31-January-07*
Number of pages:	100
EC Project Officer:	*Johan Hagman*
Keywords:	*Description Logics, Reasoning Services*
Abstract (for dissemination):	*This document describe the first version of the reasoning engine used in the BOEMIE project. Introducing representation and reasoning techniques relevant for the research issues and application issues of the BOEMIE project, the report also provide an overview about the state-of-the-art in reasoning technology based on description logics and related fields.*

Contents

Chapter 1

Introduction

The problems which application developers are facing in projects involving information technology tend to become more and more complex. In many applications – in particular in the web context – models based on large ontologies are more and more important. Moreover, aspects of distribution as well as safety and verification concerns have to be considered. In order to reduce complexity, topics such as knowledge-based systems, object-oriented frameworks, unified modeling languages, workflow models, business (re)organization models etc. dominate the discussion about how to ensure that computer systems can be constructed which, on the one hand, match the requirements and, on the other hand, are reliable, robust and adaptable. In order to fulfill these requirements there is a huge demand for systems that automatically solve complex (sub)problems of applications based on declarative models of the domain. In addition, the verification of models used for computational purposes is often required in some application scenarios. The formalization of subtasks of applications as well-understood computational inference problems is a prerequisite to support the development of more powerful systems with better quality but less development costs. It should be emphasized, however, that a user of an application need not be aware of internal computational processes being based on formal models. For instance, users can successfully interact with web-based information retrieval systems based on database technology without being aware of the underlying relational calculus.

The development of formal inference systems has a long tradition. The literature contains many contributions from different points of view and with different mathematical background. For many problems, it is appropriate that formal modeling and inference techniques are based on a logical approach, i.e. a logical characterization of various problem classes has already been developed in terms of deduction, induction, abduction, synthesis etc. In this work we pursue a computational approach combining theoretical results about the decidability of different logical representation languages with practical results concerning "efficient" proof procedures which are sound and complete (and terminating). Efficient proof techniques, in turn, are needed for implementing, for instance, query answering and specification checking systems based on expressive languages. The development of efficient algorithms that do not run into combinatorial explosion in the average case is a very active and exciting research field not only from a practical but also from a theoretical point of view.

This report describes the results of practical and theoretical investigations for developing representation systems which support the construction of declarative models, which can be checked for consistency and which can be used as the basis for problem solving processes in different application contexts. In order to support automatic processing of models, the modeling language must be based on a clear semantics such that inference problems can be formally defined, inference algorithms can be systematically analyzed, and formal inference systems can be successfully developed. Formal inference systems are of particular importance in safety-critical applications in order to check system specifications before systems are actually deployed in practice. Based on a formal model of the application domain, inference systems can also be used to directly implement subtasks of application systems. In so-called information systems, the subtasks of applications can be realized by specifying queries to be answered based on an explicitly given information model. Query answering is the basic operation found in information systems – be they web-based or not.

Usually, this scenario is the realm of information retrieval and database systems. In both research areas more and more powerful representation and query languages are investigated. Rather than being a mere information lookup, answering queries in information systems for complex domains involves reasoning about implicit information, i.e. query answering systems require inference systems as subcomponents. Thus, a new generation of information systems, especially in a web-based scenario, can be called deductive information systems.

As a challenging application of ontology-based technology, we show how formal knowledge representation and reasoning techniques can be used for the retrieval and interpretation of multimedia data. This section explains what we mean by an "interpretation" using examples of audio and video interpretation. Intuitively, interpretations are descriptions of media data at a high abstraction level, exposing interrelations and coherencies. We introduce description logics as the formal basis for ontology languages of the OWL family and for the interpretation framework described in subsequent sections.

What do we mean by "interpretation" of media objects? Consider the image shown in Figure 1.1. One can think of the image as a set of primitive objects such as persons, garbage containers, a garbage truck, a bicycle, traffic signs, trees etc. An interpretation of the image is a description which "makes sense" of these primitive objects. In our example, the interpretation could include the assertions "two workers empty garbage containers into a garbage truck" and "a mailman distributes mail" expressed in some knowledge representation language.

Figure 1.1: Street scene in Hamburg.

When including the figure caption into the interpretation process, we have a multimodal interpretation task which in this case involves visual and textual media objects. The result could be a refinement of the assertions above in terms of the location "in Hamburg" or, more specifically, "near lake Alster in Hamburg". Note that the interpretation describes activities extending in time although it is only based on a snapshot. Interpretations may generally include hypotheses about things outside the temporal and spatial scope of the available media data. An interpretation is a "high-level" description of media data in the sense that it involves terms which abstract from details at lower representation levels. This is typical for meaningful descriptions in human language and hence also relevant for constructing information systems. Information systems are one of the most-prominent application scenarios of the BOEMIE project, and the automatic ontology-based construction of high-level interpretations of media objects is one of the major tasks in this project. In this report we describe representation formalisms suitable for representing high-level image interpretation results. Based on these formalisms, inference services are described that allow for a

scalable information system architecture to be provided for building BOEMIE-based applications.

The report is structured into four parts, each with its own set of references in order to provide an overview on the state of the art for each of the areas covered. First, in Part 1, we describe research results about expressive description logics, then, in Part 2, optimization techniques for instance retrieval are discussed in the context of grounded conjunctive queries. Afterwards, in Part 3, the expressivity of the query language is enhanced in order to provide a combination of spatial and ontological reasoning. In the last part, we focus on a middleware architecture to provide efficient and scalable reasoning systems for large-scale applications.

Part I

Formalisms and Inference Problems

Chapter 2

The $\mathcal{SH}$ Family

Ontology languages like OWL, which has been adopted in BOEMIE, are often based on description logics (DLs). Since the beginning of description logic research, the tradeoff between expressivity of the logic and complexity of decision problems has been investigated in the research community [36; 37]. DLs are distinguished by the set of concept and role constructors they provide. The $\mathcal{SH}$ family is inspired by demands from applications in the sense that the language facilities of description logics of this family are designed in such a way that (i) application requirements w.r.t. expressivity can be fulfilled and (ii) reasoners can be built that are efficient in the average case. As we will see in this section, work on ontology languages based on the $\mathcal{SH}$ family revealed that there are intricate interactions between language constructs required for practical applications. Thus, reasoning gets harder (from a theoretical and from a practical point of view) if the expressivity is increased. We start our discussion of expressive description logics of the $\mathcal{SH}$ family with the core logic $\mathcal{ALC}$.

2.1 The Foundation: $\mathcal{ALC}$

The term "expressive description logic" is usually used for a logic that provides for full negation. A cornerstone logic with this feature is $\mathcal{ALC}$ [22], which is the backbone of all logics of the $\mathcal{SH}$ family. Let A be a concept name and R be a role name. Then, the set of $\mathcal{ALC}$ concepts (denoted by C or D) is inductively defined as follows: $C, D \rightarrow A \,|\, \neg C \,|\, C \sqcap D \,|\, C \sqcup D \,|\, \forall R.C \,|\, \exists R.C$. Concepts can be written in parentheses. The semantics is defined in the standard form using a Tarskian interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \text{ (complement)}$$

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \text{ (conjunction)}$$

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \text{ (disjunction)}$$

$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \,|\, \exists y.\, (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \text{ (existential restriction)}$$

$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \,|\, \forall y.\, (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \text{ (value restriction)}$$

A concept C is satisfiable if there exists an interpretation such that $C^{\mathcal{I}} \neq \emptyset$. The concept satisfiability problem of the logic $\mathcal{ALC}$ was shown to be PSPACE-complete in [111]. For $\mathcal{ALC}$ the finite model property holds.

A (generalized) terminology (also called TBox, $\mathcal{T}$) is a set of axioms of the form $C \sqsubseteq D$ (generalized concept inclusion, GCI). A GCI $C \sqsubseteq D$ is satisfied by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation which satisfies all axioms of a TBox is called a model of the TBox. A concept C is satisfiable w.r.t. a TBox if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. A concept D subsumes a concept C w.r.t. a TBox if for all models $\mathcal{I}$ of the TBox it holds that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. D is called the subsumer, C is the subsumee. A concept name A_1 mentioned in a TBox is called a most-specific

subsumer of a concept name A_2 (mentioned in the TBox and different from A_1) if A_1 subsumes A_2 and there is no other concept name A_3 (mentioned in the TBox and different from A_1 and A_2) such that A_1 subsumes A_3 and A_3 subsumes A_2. The least general subsumee of a concept name is defined analogously. The classification problem for a TBox is to find the set of most-specific subsumers of every concept name mentioned in the TBox (or knowledge base). The induced graph is called the subsumption hierarchy of the TBox.

The semantics for generalized terminological axioms (which do not impose any restrictions on the concepts on both sides of a GCI) is called descriptive semantics (see, e.g., [22] for a discussion of the consequence of this semantics and for an investigation of other possible semantics for GCIs). The problem of verifying satisfiability or checking subsumption w.r.t. generalized TBoxes is EXPTIME-hard [99; 87]. However, this result holds for the worst case, which does not necessarily occur in practical applications (see, e.g., [115]), and practical work on reasoners for languages of the $\mathcal{SH}$ family exploits this insight.

As of now, we have covered axioms for expressing restrictions for *subsets* of the domain $\Delta^{\mathcal{I}}$ (or subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). For BOEMIE, restrictions about particular elements of the domain are also important, since it is of interest to be able to classify the multimodal enquired information according to a generalized terminology of the sports domain. For this task, early description logics of the $\mathcal{SH}$ family provide specific assertions that are collected in a so-called ABox. An ABox is a set of assertions of the form $C(a)$, $R(a, b)$, $a = b$, or $a \neq b$. An ABox assertion is satisfied by an interpretation $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, $a^{\mathcal{I}} = b^{\mathcal{I}}$, and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$, respectively. The ABox satisfiability problem (w.r.t. a TBox) is to check whether there exists an interpretation (a model of the TBox) that satisfies all ABox assertions. As usual, we define a knowledge base (KB) to be a pair $(\mathcal{T}, \mathcal{A})$ of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A model of a KB is an interpretation that is a model of $\mathcal{T}$ and $\mathcal{A}$. The instance problem $instance_{(\mathcal{T},\mathcal{A})}(i, C)$ w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ is to test whether $i^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models of the knowledge base. We say $instance_{(\mathcal{T},\mathcal{A})}(i, C)$ is entailed. The knowledge base is often omitted in the notation if clear from context. The instance retrieval problem $retrieve_{(\mathcal{T},\mathcal{A})}(C)$ w.r.t. a KB $(\mathcal{T}, \mathcal{A})$ and a query concept C is to determine all individuals i mentioned in the ABox for which $instance(i, C)$ is entailed. A role filler for a role R w.r.t. an individual i is an individual j (mentioned in the ABox) such that for all models $\mathcal{I}$ it holds that $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ (we say $related(i, j, R)$ is entailed).

The inference problems mentioned in this section are called standard inference problems for TBoxes and ABoxes, respectively. Reasoners of the $\mathcal{SH}$ family support standard inference problems either for TBoxes and ABoxes or for TBoxes only. As we have seen, $\mathcal{ALC}$ inference problems are not tractable in the worst case, and, at the beginning, incomplete algorithms were used in concrete system implementations for expressive DLs. However, at the end of the eighties it became clear that incomplete algorithms for expressive description logics cause all kinds of problems for applications. For instance, more often than not, the addition of an axiom or assertion to the knowledge base led to the situation that previously obtained entailments were no longer computed due to peculiarities of the inference algorithm.

The beginning of the $\mathcal{SH}$ family started with work on the system KRIS [10; 57; 1], which provides a sound and complete reasoner based on the tableau calculus presented in [111]. KRIS supports $\mathcal{ALC}$ plus number restrictions (plus some additional language constructs). The KRIS reasoner implements optimization techniques for the concept and ABox satisfiability problem w.r.t. TBoxes (e.g., lazy unfolding, trace technique). The main achievement of this work is that the architecture of KRIS is tailored towards specific services for TBoxes, namely TBox classification. Specific optimization techniques for the classification problem developed for KRIS are used by all contemporary reasoners of the $\mathcal{SH}$ family (see below). The idea is to classify a TBox using a top-down and bottom-up search phase for computing the most-specific subsumers and least-specific subsumees based on subsumption tests. KRIS avoids unnecessary subsumption tests using marker propagation tests [12; 7].

2.2 Concrete Domains

Another achievement of the work on description logics that is also important for ontology languages is the treatment of specific domains with fixed (concrete) semantics. To denote for instance that

Running_60_Meters is a running modality conducted on a 60 meters track ($Running\_60\_Meters \sqcap \exists meters \leq 60$), applications require constraints over the reals, the integers, or a domain of strings. A concrete domain $\mathcal{D}$ is a tuple $(\Delta^{\mathcal{D}}, \Phi)$ of a non-empty set $\Delta^{\mathcal{D}}$ and a set of predicates Φ. Predicates are defined in a certain language (e.g., linear inequations over polynomials or equations over strings). The integration of concrete domains into $\mathcal{ALC}$ is investigated in [8; 9]. The idea of the new language, $\mathcal{ALC}(\mathcal{D})$, is that the axioms for capturing the concrete semantics of the objects in $\Delta^{\mathcal{D}}$ is not captured with description logic axioms but somehow represented separately. The tableau calculus in [8; 9] treats the satisfiability problem w.r.t. to conjunctions of concrete domain predicates as separate subproblems. The concrete domain satisfiability problems must be decidable (admissibility criterion for concrete domains).

With concrete domains, so-called attributes are introduced, which are partial functions that map individuals of the abstract domain $\Delta^{\mathcal{I}}$ to elements of $\Delta^{\mathcal{D}}$ of the concrete domain $\mathcal{D}$. For attributes a, the interpretation is extended as follows: $a^{\mathcal{I}} : \Delta^{\mathcal{I}} \longrightarrow \Delta^{\mathcal{D}}$.

It is important to note that in the original approach [8; 9] it is possible to relate (multiple) attribute values of *different* individuals of the domain $\mathcal{I}$. One can represent, for instance, structures such as lists of numbers with decreasing value where each value is at most half as large as the predecessor. If the language provides concrete domains such as, for instance, linear inequations over the reals, GCIs cannot be supported by a description logic of interesting expressivity part due to undecidability of major inference problems. This follows from a result in [86] (a direct proof was developed at the same time and is given in [94]). In a restricted form where no feature compositions can be used, it is only possible to relate attribute values of a single element of $\mathcal{I}$. We use the notation $\mathcal{ALC}(\mathcal{D})^-$ to indicate that feature chains are omitted. Concrete domains are part of many specific description logics of the $\mathcal{SH}$ family that we cover in the next sections.

2.3 Transitive Roles

For many applications, part-whole relations are important. A characteristic of some part-whole relations is that they are transitive (see, e.g., [80]). In order to cope with these modeling demands, for instance, in process engineering applications, an investigation about an extensions of $\mathcal{ALC}$ with means to express transitivity was carried out [108; 109]. $\mathcal{ALC}$ was extended with a transitive closure operator, with transitive roles, and with so-called transitive orbits. As discussed in other sections, $\mathcal{ALC}$ extended with a transitive closure operator causes the concept satisfiability problem to move from PSPACE to EXPTIME.

Syntactically, transitive roles are indicated as a subset of all role names. It turned out that $\mathcal{ALC}$ extended with transitive roles remains in PSPACE [108]. Transitive roles have the semantics that for all transitive roles R the models must satisfy $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$. Thus, transitive roles are "globally" transitive and cannot be used in a transitive way in a local setting only (as possible with a specific operator for the transitive closure of a role).

Inspired by work on modal logics, [108] introduces an elegant way to integrate reasoning about transitive roles into the $\mathcal{ALC}$ tableau calculus by a special rule for transitive roles in value restrictions. Additionally, in order to enforce termination, *blocking conditions* were defined such that the calculus terminates. A blocking condition involves a test whether two sets of concepts are in a certain relation (for $\mathcal{ALC}_{R^+}$, the relation is $\subseteq$, for details see [108]).

The logic was initially called $\mathcal{ALC}_{R^+}$. As more language constructs were added later on, and acronyms became hard to read, $\mathcal{ALC}_{R^+}$ was renamed $\mathcal{S}$ in [69].[1]

2.4 Role Hierarchies and Functional Restrictions

Inspired by work on medical domains in which it became important to represent that some relations are subrelations (subsets) of other relations, so-called role inclusions axioms of the form $R \sqsubseteq S$ (with R and S being role names) were investigated in [61] as an extension to $\mathcal{ALC}_{R^+}$. A set of role inclusion axioms is called a role hierarchy. Models for role hierarchies are restricted to satisfy $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for all $R \sqsubseteq S$. The description logic is called $\mathcal{ALCH}_{R^+}$ or $\mathcal{SH}$.

[1] The name is inspired by modal logic $\mathbf{S4_m}$ but, obviously, it is a misnomer. However the name is kept for historical reasons.

Role hierarchies introduce explicit names for so-called subroles. In [62] it is argued that role hierarchies provide for adequate expressivity while still allowing for efficient practical implementations at the same time. Another possibility would have been to consider a role-forming operator for constructing role conjunctions ($R \sqcap S$). However, except for inverse roles (see below) the $\mathcal{SH}$ family includes no role-forming operators in order to provide for practically efficient implementations (see also the discussion about a transitive closure operator for roles in the previous subsection).

Additionally, in [61; 62] global functional restrictions on roles were investigated. In the corresponding description logic $\mathcal{ALCH}f_{R^+}$ so-called features were introduced as a specific subset of the role names.[2] Features must not be transitive. The semantics of a feature f is a (single valued) partial function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \longrightarrow \Delta^{\mathcal{I}}$.

With several examples, the interactions of role hierarchies and functional restrictions on roles were demonstrated in [61]. A sound and complete tableau calculus for $\mathcal{ALCH}f_{R^+}$ is described in [63]. This tableau calculus provided the basis for the enormous success of the $\mathcal{SH}$ family of ontology languages. Based on an optimized implementation of this calculus in the system FaCT [61; 62] it was shown that description logics could provide a solid basis for practical application of ontology languages. Role hierarchies and transitive roles allow one to somehow "simulate" GCIs (by constructing an equisatisfiable knowledge base). However, the FaCT system also included full support for GCIs.

The contribution of the $\mathcal{ALCH}f_{R^+}$ reasoner FaCT is (at least) threefold. First, improvements to propositional satisfiability search algorithms [41] were incorporated into description logic systems (backjumping, boolean constraint propagation, semantic branching, etc.) and, second, classification operations were dramatically increased by the invention of a so-called model merging operation [61], which exploits that most subsumption tests for concept names A_1 and A_2 (used to compute the subsumption hierarchy) return false. The idea of a model merging operation is to compute (small) data structures for concept names (and their negations) such that it is more or less directly obvious that the conjunction $A_1 \sqcap \neg A_2$ is satisfiable (i.e., there is no subsumption relation). Third, using algebraic transformation, FaCT showed that, in many practical applications, corresponding TBox axioms can be converted into a form such that lazy unfolding is maximally exploited in the tableau calculus (GCI absorption [72]). The system FaCT initiated the breakthrough of description logics as the basis for practically used ontology languages. FaCT was designed for TBoxes only.

2.5 Number Restrictions and Inverse Roles

The need for restrictions on the number of role fillers of an individual, to express for instance that a first place athlete is one which has at least 1 gold medal ($\exists_{\geq 1} has\_medal.GoldMedal$) are also apparent. Number restrictions are concept construction operators of the form ($\exists_{\leq n} R$) or ($\exists_{\geq n} R$) (simple number restrictions, indicated with letter $\mathcal{N}$ in language names) and ($\exists_{\leq n} R.C$) or ($\exists_{\geq n} R.C$) (qualified number restrictions [59], indicated with letter $\mathcal{Q}$ in language names). For simple number restrictions, interpretations must satisfy $(\exists_{\leq n} R)^{\mathcal{I}} = \{x \mid \|\{y|(x,y) \in R^{\mathcal{I}}\}\| \leq n\}$ and $(\exists_{\geq n} R)^{\mathcal{I}} = \{x \mid \|\{y|(x,y) \in R^{\mathcal{I}}\}\| \geq n\}$. For qualified number restrictions, interpretations must satisfy $(\leq n\, R.C)^{\mathcal{I}} = \{x \mid \|\{y|(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\| \leq n\}$ and $(\geq n\, R.C)^{\mathcal{I}} = \{x \mid \|\{y|(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\| \geq n\}$.

KRIS supported simple number restrictions in a system implementation at the end of the eighties. With only simple number restrictions and no role inclusions, it is possible to use a single placeholder for an arbitrarily large number of role fillers whose existence required by a number restriction (see [6] for details). Results on the interaction of number restrictions and role conjunctions were developed with $\mathcal{ALCNR}$ [30; 31]. Simple reasoning with placeholders is no longer possible with these operators being part of the language. The same holds for number restrictions in combination with role hierarchies as used in the $\mathcal{SH}$ family.

In addition to problems w.r.t. placeholder reasoning in the presence of number restrictions, it was shown that there is a strong interaction between number restrictions and transitive roles (and role hierarchies). Allowing number restrictions with transitive roles (or roles which have transitive subroles) leads to undecidability [69]. As a consequence, so-called *simple* roles were introduced

[2] Note that $\mathcal{ALCH}f_{R^+}$ does not provide role-value maps as supported by $\mathcal{ALCF}$ [60; 58].

into the $\mathcal{SH}$ family. In (qualified) number restrictions, only simple roles are allowed. With this restriction, inference problems become decidable [69].

Another demand from practical applications was the support for inverse roles (letter $\mathcal{I}$ in language names). In [69] the research on a corresponding role-forming operator $\cdot^{-1}$ in the context of the $\mathcal{SH}$ family is summarized. Again, a subtle interaction between number restrictions (or features), inverse roles as well as transitive roles and role hierarchies (or GCIs) was discovered. If all these constructs are used, the finite model property does no longer hold. First, due to inverse roles, the trace technique is no longer applicable, and, second, the application of the blocking condition introduced in the work about $\mathcal{ALC}_{R^+}$ had to be made considerably more complex. Blocking must be dynamic [69]. This makes the implementation of concrete reasoning systems much more difficult. An additional source of complexity is that blocking must not occur too early (thus, the blocking condition involves a test for set equality), and, furthermore, due to infinite models, the blocking condition involves a pair of two sets of concepts (pairwise blocking [69]).

Although ABoxes were also investigated in the context of $\mathcal{ALCNR}$, work on the $\mathcal{SH}$ family of description logic languages initially considered TBoxes only.

2.6 Number Restrictions, ABoxes, and Concrete Domains

Inspired by work on the $\mathcal{SH}$ family and work on ABoxes in $\mathcal{ALCNR}$ as well as work on concrete domains [8], a tableau calculus for ABox reasoning in the language $\mathcal{ALCNH}_{R^+}$ was presented in [47] and concrete domain reasoning was investigated in this context in [54]. The insights of this work are that in the presence of ABoxes, (i) models are no longer (quasi) trees but forests, (ii) individuals mentioned in the ABox must not block each another, and (iii) on the concrete domain part of the language, feature chains cannot be supported (for all kinds of concrete domains) in order to preserve decidability. A tableau calculus for reasoning about ABoxes in the language $\mathcal{SHIQ}$ (aka $\mathcal{ALCQHI}_{R^+}$) with ABoxes was presented shortly afterwards in [70] (but concrete domains were not considered).

The latter work led to a new version of the FaCT system (iFaCT) for supporting TBoxes with inverse roles. The above-mentioned research contributions also provided the basis for the implementation of the RACER reasoner [51], a DL system for TBoxes and ABoxes with concrete domains for linear inequations over the reals and the cardinals as well as inequations over strings and booleans. First versions of both systems, iFaCT and RACER, appeared at the end of the nineties, i.e. both systems support the language $\mathcal{SHIQ}$ [3] (see Figure 2.1 for syntax and semantics). Both RACER and FaCT use the TBox classification techniques developed for KRIS [12; 7].

Optimized reasoning techniques for $\mathcal{SHIQ}$ w.r.t. blocking [66] were developed for later versions of the iFaCT system, and also included in RACER. The idea is to relax the blocking condition for inverse roles (see above) and retain the subset tests for some parts of the concept set involved in the blocking test (see [66] for details).

With RACER, optimized reasoning for qualifying number restrictions [50; 48] was investigated. The work is based on [104].

Due to the continuous semantics extraction and ontology evolution process in BOEMIE, it is necessary to be able to cope with huge amounts of data, which is continuously increasing, giving rise to a large number of concept names. Thus, in order to classify huge terminologies with RACER, a refinement of the techniques introduced in [12; 7] is presented in [49]. Topological sorting of transformed GCIs to classify concepts in definition order allows to skip the bottom-up search phase. Optimizations for concrete domains in terms of extended model merging operations and incremental concrete domain satisfiability testing during a tableau proof are described in [53]. GCI absorption strategies are also investigated with RACER, e.g., absorption of domain and range restrictions (see also [124] for similar techniques in FaCT).

Concrete domain reasoning is still actively explored. Starting with investigation involving new combination operators ([88]), in [89; 90] it is shown that for specific concrete domains, feature chains can indeed be allowed in the presence of GCIs (see also [91; 92]). The language investigated (Q-$\mathcal{SHIQ}$) provides predicates for linear inequalities between variables (but no polynomials). A

[3] In RACER initially the unique name assumption was always employed, in later versions the assumption could be activated on demand.

(a)

Syntax	Semantics
Concepts ($\mathsf{R} \in R$, $\mathsf{S} \in S$, and $\mathsf{f}, \mathsf{f_i} \in F$)	
A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ (A *is a concept name*)
$\neg\mathsf{C}$	$\Delta^{\mathcal{I}} \setminus \mathsf{C}^{\mathcal{I}}$
$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}}$
$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cup \mathsf{D}^{\mathcal{I}}$
$\exists\,\mathsf{R}\,.\,\mathsf{C}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \exists\,\mathsf{b} \in \Delta^{\mathcal{I}} : (\mathsf{a},\mathsf{b}) \in \mathsf{R}^{\mathcal{I}} \wedge \mathsf{b} \in \mathsf{C}^{\mathcal{I}}\}$
$\forall\,\mathsf{R}\,.\,\mathsf{C}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \forall\,\mathsf{b} \in \Delta^{\mathcal{I}} : (\mathsf{a},\mathsf{b}) \in \mathsf{R}^{\mathcal{I}} \Rightarrow \mathsf{b} \in \mathsf{C}^{\mathcal{I}}\}$
$\exists_{\geq n}\,\mathsf{S}\,.\,\mathsf{C}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \|\{\mathsf{x} \mid (\mathsf{a},\mathsf{x}) \in \mathsf{S}^{\mathcal{I}}, \mathsf{x} \in \mathsf{C}^{\mathcal{I}}\}\| \geq \mathsf{n}\}$
$\exists_{\leq m}\,\mathsf{S}\,.\,\mathsf{C}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \|\{\mathsf{x} \mid (\mathsf{a},\mathsf{x}) \in \mathsf{S}^{\mathcal{I}}, \mathsf{x} \in \mathsf{C}^{\mathcal{I}}\}\| \leq \mathsf{m}\}$
$\exists\,\mathsf{f_1},\ldots,\mathsf{f_n}\,.\,\mathsf{P}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \exists\,\mathsf{x_1},\ldots,\mathsf{x_n} \in \Delta^{\mathcal{D}} : (\mathsf{a},\mathsf{x_1}) \in \mathsf{f_1}^{\mathcal{I}} \wedge \ldots \wedge (\mathsf{a},\mathsf{x_n}) \in \mathsf{f_n}^{\mathcal{I}} \wedge (\mathsf{x_1},\ldots,\mathsf{x_n}) \in \mathsf{P}^{\mathcal{I}}\}$
$\forall\,\mathsf{f_1},\ldots,\mathsf{f_n}\,.\,\mathsf{P}$	$\{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \forall\,\mathsf{x_1},\ldots,\mathsf{x_n} \in \Delta^{\mathcal{D}} : (\mathsf{a},\mathsf{x_1}) \in \mathsf{f_1}^{\mathcal{I}} \wedge \ldots \wedge (\mathsf{a},\mathsf{x_n}) \in \mathsf{f_n}^{\mathcal{I}} \Rightarrow (\mathsf{x_1},\ldots,\mathsf{x_n}) \in \mathsf{P}^{\mathcal{I}}\}$
Roles and Features	
R	$\mathsf{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
f	$\mathsf{f}^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{D}}$ (features are partial functions)

$\|\cdot\|$ denotes the cardinality of a set, and $n, m \in \mathbb{N}$ with $n > 1$, $m > 0$.

(b)

Axioms	
Syntax	Satisfied if
$\mathsf{R} \in T$	$\mathsf{R}^{\mathcal{I}} = (\mathsf{R}^{\mathcal{I}})^{+}$
$\mathsf{R} \sqsubseteq \mathsf{S}$	$\mathsf{R}^{\mathcal{I}} \subseteq \mathsf{S}^{\mathcal{I}}$
$\mathsf{C} \sqsubseteq \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$

(c)

Assertions ($\mathsf{a},\mathsf{b} \in O$, $\mathsf{x},\mathsf{x_i} \in O_C$)	
Syntax	Satisfied if
$\mathsf{a}:\mathsf{C}$	$\mathsf{a}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$
$(\mathsf{a},\mathsf{b}):\mathsf{R}$	$(\mathsf{a}^{\mathcal{I}},\mathsf{b}^{\mathcal{I}}) \in \mathsf{R}^{\mathcal{I}}$
$(\mathsf{a},\mathsf{x}):\mathsf{f}$	$(\mathsf{a}^{\mathcal{I}},\alpha(\mathsf{x})) \in \mathsf{f}^{\mathcal{I}}$
$(\mathsf{x_1},\ldots,\mathsf{x_n}):\mathsf{P}$	$(\alpha(\mathsf{x_1}),\ldots,\alpha(\mathsf{x_n})) \in \mathsf{P}^{\mathcal{I}}$
$\mathsf{a} = \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} = \mathsf{b}^{\mathcal{I}}$
$\mathsf{a} \neq \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} \neq \mathsf{b}^{\mathcal{I}}$

Figure 2.1: Syntax and Semantics of $\mathcal{ALCQHI}_{R^+}$.

more modular approach is described in [93; 83] where the notion of admissibility (see above) is extended to a so-called ω-admissibility. No system implementation exists at the time of this writing.

Reasoning systems for the $\mathcal{SH}$ family are successful because of research on average-case behavior and appropriate optimization techniques. These systems analyse the language of the input problem and select appropriate optimizations to answer queries as fast as possible, moreover, they are based on sound and complete algorithms.

Optimizations for instance retrieval w.r.t. ABoxes is investigated in [52]. An important property of the $\mathcal{SHIQ}$ language is that the subsumption hierarchy of the TBox part of a knowledge base $(\mathcal{T},\mathcal{A})$ is stable w.r.t. additions to the ABox part. Stability means that the subsumption relation between concepts C and D depends only on axioms in $\mathcal{T}$. This property is exploited in practical ABox systems such as RACER (and also older systems such as KRIS). Multiple knowledge bases $(\mathcal{T},\mathcal{A}_1)$, ..., $(\mathcal{T},\mathcal{A}_k)$ with the same TBox $\mathcal{T}$ can be efficiently supported in the sense that computations for the TBox can be reused for answering queries on any of the ABoxes $\mathcal{A}_i$. Unfortunately, the stability property is lost with the introduction of cardinalities for concepts or with the inclusion of so-called nominals, which became necessary in order to further increase the expressivity of $\mathcal{SHIQ}$ for some applications.

2.7 Application Example: Description of Media Objects

An application scenario for automatically derived interpretations of media objects is information retrieval, for instance, in the semantic web. Interpretations are seen as annotations of media objects and can be practically represented in RDF or OWL format. In our view, annotations describe "real-world" objects and events. It is not the goal to merely "classify" images and attach keywords but to construct a high-level interpretation of the content of a media object. The former approach has a limited applicability if examples such as Figure 1.1 are considerd and query for, e.g., media objects with a mailman have to be answered.

A set of media objects with annotations attached to each media object can be made available via

a web server with standard application server technology. We assume that the web server provides a query interface (for instance, using the XML-based DIG 2.0 or OWL-QL query language, see Section 3.2). For readability reasons, however, here we use ABoxes for RDF descriptions, and employ a mathematical notation for queries. Using the example from Figure 1.1 we sketch how media interpretations are used to implement an media retrieval system.

$$
\begin{array}{rcl}
mailman_1 & : & Mailman \\
bicycle_1 & : & Bicycle \\
mail\_deliv_1 & : & MailDelivery \\
(mail\_deliv_1, mailman_1) & : & hasPart \\
(mail\_deliv_1, bicycle_1) & : & hasPart \\
(mail\_deliv_1, url\_1) & : & hasURL \\
(mailman_1, url\_2) & : & hasURL \\
(bicycle_1, url\_3) & : & hasURL \\
(url\_1) & : & =\texttt{"http://www.img.de/image-1.jpg"} \\
(url\_2) & : & =\texttt{"http://www.img.de/image-1.jpg\#(200,400)/(300/500)"} \\
(url\_3) & : & =\texttt{"http://www.img.de/image-1.jpg\#(100,400)/(150/500)"} \\
garbageman_1 & : & Garbageman \\
garbageman_2 & : & Garbageman \\
garbagetruck_1 & : & Garbage\_Truck \\
garbage\_coll_1 & : & Garbage\_Collection \\
(garbage\_coll_1, garbageman_1) & : & hasPart \\
(garbage\_coll_1, garbageman_2) & : & hasPart \\
(garbage\_coll_1, garbagetruck_1) & : & hasPart \\
(garbage\_coll_1, url\_4) & : & hasURL \\
\multicolumn{3}{c}{\ldots}
\end{array}
$$

Figure 2.2: An ABox representing the annotation of the image in Figure 1.1. The predicate $=_{string}$ stands for a one-place predicate $p(x)$ which is true for $x = string$.

The example in Figure 2.2 illustrates the main ideas about annotations for media objects using ABoxes (we omit the TBox for brevity). It would have been possible to more appropriately describe the role which the parts play in the events (in the sense of case frames). We omit the discussion of these issues here for brevity, however, and use a "generic" role $hasPart$. It is also possible to use another "agggreate" $street\_scene_1$ for combining the garbage collection and mail delivery events.

With axioms such as

$$Mailman \sqsubseteq Postal\_Employee$$

$$Mailman \equiv Postman$$

queries can be formulated w.r.t. different vocabularies. Inference services such as query answering are formally defined in the next chapter. Before describing details of the definition of inference problems we shortly sketch recent work about modeling uncertainty and vagueness in a description logic context.

2.8 Additional Representation Techniques

The operators used in the example above are provided by state-of-the-art reasoning systems (see below for an overview on systems). Some additional operators for concept and roles as well as additional types of axioms have been investigated in the literature.

2.8.1 Nominals

A nominal denotes a singleton concept. The syntax is $\{o\}$ and the semantics w.r.t. the interpretation is $\{o\}^{\mathcal{I}} = \{o^{\mathcal{I}}\}$. With nominals it is possible to relate all individuals of a certain concept to a particular individual (e.g., all athletes that come from a particular country called Italy). Nominals

were first investigated in [110] and are related to cardinality restrictions on concepts [11; 5]. The first system with support for nominals was CRACK [29].

Although nominals in the context of $\mathcal{SHIQ}$ were proven to be decidable (see [122]) it took some time until the first tableau calculus was presented for the language $\mathcal{SHOQ}(\mathcal{D})$ [65]. This work also introduced so-called datatype roles [65], which must not be confused with concrete domain attributes. Datatype roles map objects from the domain to *sets of* objects from a concrete domain. In $\mathcal{SHOQ}(\mathcal{D})$ concrete domain predicates apply to (multiple) datatype properties of a single object of the interpretation domain $\Delta^{\mathcal{I}}$. It is *not* possible to enforce constraints on datatype values of *multiple* objects from $\Delta^{\mathcal{I}}$. This insight gives rise to corresponding optimization techniques but it should be noted that some expressivity is lost.

The distinction between TBoxes and ABoxes is no longer required for languages with nominals. Instead of using $C(a)$ or $R(a, b)$ as ABox assertion one can just write GCIs such as $\{a\} \sqsubseteq C$ or $\{a\} \sqsubseteq \exists R.\{b\}$, respectively.[4] Even if ABoxes would be supported by practical systems, it is obvious that the subsumption relation is not stable for languages with nominals.

Intricate interactions of nominals in $\mathcal{SHOQ}(\mathcal{D})$ with inverse roles were investigated in $\mathcal{SHOIQ}(\mathcal{D})$ [67]. Indeed, it was shown that concept satisfiability in $\mathcal{SHOIQ}(\mathcal{D})$ is NEXPTIME-complete.

2.8.2 Acyclic Role Axioms

Further results on optimized classification [126; 125] has opened up additional application areas for ontology languages. And, although much has been achieved by dedicated optimization techniques developed for the $\mathcal{SH}$ family of description logic languages, still there are hard knowledge bases known (e.g., [25]). New language features with respect to specific kinds of role axioms involving role composition have been proposed for medical domains. A tableau calculus for the new language $\mathcal{SROIQ}(\mathcal{D})$ is presented in [64] proving that it is decidable. To the best of our knowledge, there is no system implementation at the time of this writing, that supports all features of this language which is the most expressive language w.r.t to role statements. On top of $\mathcal{SHIQ}$, $\mathcal{SROIQ}(\mathcal{D})$ allows for more expressivity concerning roles, where besides TBox and ABox, an RBox is introduced to include role statements, allowing for:

1. *Complex role inclusion axioms* of the form $R \circ S \sqsubseteq R$ and $S \circ R \sqsubseteq R$ where R is a role and S is a simple role.
2. *Disjoint roles*
3. *Reflexive, irreflexive and antisymmetric roles*
4. *Negated role assertions*
5. *Universal roles*
6. *Local expressivity* to allow concepts of the form $\exists R.\text{Self}$

$\mathcal{SROIQ}(\mathcal{D})$ represents the logical basis of OWL 1.1 plus datatypes and datatypes restrictions.

2.8.3 Integration of DLs and rule languages

Decidability is a characteristic that should be preserved by ontology languages and which has caused expressivity restrictions. This is one of the reasons why rules are gaining interest as an option to overcome expressivity limitations in DLs. A relevant proposal to extend DL languages (more specifically, the syntactic variant OWL-DL) with Horn-like rules, is the rule language called SWRL (Semantic Web Rule Language). SWRL uses OWL DL or OWL Lite as the underlying DL language to specify a KB.

But the extension of OWL DL with rules is known to be undecidable [98], this is due to the fact that decidability in OWL DL restricts the language to axioms that expresses only quasi tree-like structures (we disregard transitive relations). Such a property is lost when adding rules, therefore in order to add rules and still preserve decidability, a subset of SWRL can be used, the so called *DL-safe* rules [98].

[4] Equality and inequality of individuals can also easily be specified using negation.

DL-safe rules are applicable only to explicitly introduced individuals and are formally defined as follows:

Given a set of concept names N_c, a set of abstract and concrete role names $N_{R_a} \cup N_{R_c}$. A DL-atom is of the form C(x) or R(x,y), where $C \subseteq N_C$ and $R \in N_{R_a} \cup N_{R_c}$. A rule r is safe if each of its variables occurs in a non-DL-atom in the rule body. In practice this means that rules are applied to ABox individuals only.

2.8.4 Modeling Uncertainty and Vagueness in a Description Logic Context

Modeling uncertainty in the context of description logics (DL) has been a topic of research since many years. An overview of such extensions to classical description logics is presented in [21]. The research is oriented to the work of modeling uncertain knowledge on the basis of first-order structures [100; 23; 55]. The fundamental view of the approaches based on description logics, is such that, it should also be possible to represent the degree of overlap between concepts (and not only subsumption or disjunction) through probabilities. Furthermore it should also be possible to formulate uncertainty about the structure of objects. Initial approaches considered primarily probabilistic knowledge at the conceptual level, this means, at the level of the TBox (e.g., [56]). Also knowledge representation for single objects and their relations from a probabilistic view were studied [74], such that structural uncertainty could potentially be modeled. Along with early research results about decidability of very expressive logics (e.g., OWL DL), proposals for the modeling of uncertain knowledge were given.
In [44], a probabilistic description logic language was studied, in which it is possible to formulate in addition to probabilistic knowledge at the conceptual level (i.e., TBox), also assertional probabilistic knowledge (i.e., ABbox) about concepts and role instances. In this language (P-$\mathcal{SHOQ}$) there is no more a separation between TBox and ABox for the modeling of uncertainty. Its underlying reasoning formalism is based on probabilistic lexicographic entailment by [81]. Lexicographic entailment is based in default logic and makes use of model creation to look for preferred minimal models, where the minimal verifying (resp., falsifying) model determines entailment (resp., not entailment). In [44] the work of [81] is extended from a propositional logic to a first-order logic, furthermore [44] generalizes classical interpretations to probabilistic interpretations by adding a probability distribution over the abstract domain and by interpreting defaults as statements of high conditional probability. E.g., in [81] a default like $P(bird \rightarrow fly) \geq 1 - \varepsilon$ is in [44] a conditional constraint like $l \leq P(fly|bird) \leq u$. The work of [44] allows to represent probabilistic knowledge in a description logic language with high expressivity.
It is important to observe that the semantics used in the different approaches do not differ much (for example w.r.t. [74] and [44]). An approach for the modeling of uncertain structures for a less expressive language is presented in [38]. However, no specific inference algorithms are known for this approach. An important step for the practical use of description logics with probabilities occurred with the integration of Bayesian networks in P-CLASSIC [79], nevertheless very strong disadvantages were obtained: For number restrictions the supremum limits must be known and separate Bayesian networks are necessary to consider role fillers. Along with this problem, the probabilistic dependencies between instances must also be modeled. This problem was overcome in [78] - however not in the context of description logics but with a frame-based approach, in which the treatment of default values is given without formal semantics. The main idea in [78] is the view of considering role fillers as nodes in Bayesian networks which have CPTs (conditional probability tables) associated to them as generalized number restrictions in the sense of description logics. This view gives an important basis for our project. Related studies are followed in [106].

Complementary to the P-CLASSIC approach, another approach [131](PTDL) was developed for probabilistic modeling with the use of first-order structures. In this approach the Bayesian network theory is considered as basis reference for further extensions, instead of (classical) description logics. The Bayesian network nodes represent function values and an individual is associated to other nodes through these function values. The approach in [131] avoids some disadvantages of P-CLASSIC, but it offers minimal expressivity on the side of description logics. In context with very expressive description logics another approach [34; 35] was presented for the integration of Bayes networks. Algorithms for deduction over probabilistic first-order structures was developed by Poole

[107]. Poole observes, that the existing approaches (e.g., [78; 106]) only consider individuals that are explicitly named. Qualitative probabilistic matching with hierarchical descriptions was also studied [114] and allows the variation of the level of abstraction. Previous studies about the dependencies between the theories used in databases relates to Datalog and description logics developed in the artificial intelligence area (the so called description logic programs), influenced the probabilistic approaches implemented for Datalog could also be transferred to probabilistic approaches on description logics.[102; 84; 85; 103]. Approaches for information retrieval with probabilistic Datalog are presented in [43; 42]. In this area, work on learning from Datalog-predicates with uncertainty is also relevant [101].
Modeling vagueness to capture notions of imprecise knowledge has been intensively studied [121; 123; 132], such that existing knowledge representation formalisms like first-order logic can be extended to represent vague concepts, (e.g., hot, cold) which are not entirely true or false, but rather have a truth value between true and false. Fuzzy Logic, with a basis in fuzzy set theory, allows the modelling of vagueness, and its fundamental view is that the classical ideas of satisfiability and subsumption are modified such that concepts are satisfiable to a certain degree, or two concepts subsume to a certain degree. In [123] a tableau-like method for computing the degree of subsumption between two concepts in the language $\mathcal{ALC}_{fm}$ was presented. In [132] work on extending description logics with fuzzy features is presented for the language FL^-, in which it is possible to determine subsumption, but not possible to determine whether a individual is an instance of a concept with a certain probability. In [121], the use of fuzzy logic is highlighted in the context of multimedia information retrieval, in which images are semantically annotated with fuzzy statements.

Recently, more expressive fuzzy description logics have been investigated [118; 119; 117; 105; 120; 116].

Chapter 3

Inference Services

3.1 Standard Inference Services

In the following we define standard inference services for description logics.

A *concept* is called *consistent* (w.r.t. a TBox $\mathcal{T}$) if there exists a model of C (that is also a model of $\mathcal{T}$). An *ABox* $\mathcal{A}$ is *consistent* (w.r.t. a TBox $\mathcal{T}$) if $\mathcal{A}$ has model $\mathcal{I}$ (which is also a model of $\mathcal{T}$). A *knowledge base* ($\mathcal{T}$,$\mathcal{A}$) is called *consistent* if there exists a model for $\mathcal{A}$ which is also a model for $\mathcal{T}$. A concept, ABox, or knowledge base that is not consistent is called *inconsistent*.

A concept D *subsumes* a concept C (w.r.t. a TBox $\mathcal{T}$) if $\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ (that are models of $\mathcal{T}$). If D subsumes C, then C is said to be *subsumed by* D.

Besides these basic problems, some additional inference services are provided by description logic systems. A basic reasoning service is to compute the subsumption relationship between concept names. This inference is needed to build a hierarchy of concept names w.r.t. specificity. The problem of computing the most-specific concept names mentioned in $\mathcal{T}$ that subsume a certain concept is known as computing the *parents* of a concept. The *children* are the most-general concept names mentioned in $\mathcal{T}$ that are subsumed by a certain concept. We use the name *concept ancestors* (*concept descendants*) for the transitive closure of the parents (children) relation. The computation of the parents and children of every concept name is also called *classification* of the TBox. Another important inference service for practical knowledge representation is to check whether a certain concept name occcurring in a TBox is inconsistent. Usually, inconsistent concept names are the consequence of modeling errors. Checking the consistency of all concept names mentioned in a TBox without computing the parents and children is called a TBox *coherence check*.

If the description logic supports full negation, consistency and subsumption can be mutually reduced to each other since D subsumes C (w.r.t. a TBox $\mathcal{T}$) iff $\mathsf{C} \sqcap \neg\mathsf{D}$ is inconsistent (w.r.t. $\mathcal{T}$) and C is inconsistent (w.r.t. $\mathcal{T}$) iff C is subsumed by $\bot$ (w.r.t. $\mathcal{T}$). Consistency of concepts can be reduced to ABox consistency as follows: A concept C is consistent (w.r.t. a TBox $\mathcal{T}$) iff the ABox $\{\mathsf{a}:\mathsf{C}\}$ is consistent (w.r.t. $\mathcal{T}$).

An individual i is an *instance* of a concept C (w.r.t. a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$) iff $\mathsf{i}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$ for all models $\mathcal{I}$ (of $\mathcal{T}$ and $\mathcal{A}$). For description logics that support full negation for concepts, the instance problem can be reduced to the problem of deciding if the ABox $\mathcal{A} \cup \{\mathsf{i}:\neg\mathsf{C}\}$ is inconsistent (w.r.t. $\mathcal{T}$). This test is also called *instance checking*. The most-specific concept names mentioned in a TBox $\mathcal{T}$ that an individual is an instance of are called the *direct types* of the individual w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$. The direct type inference problem can be reduced to subsequent instance problems (see e.g. [7] for details).

An ABox $\mathcal{A}'$ is *entailed* by a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ if all models of $\mathcal{T}$ and $\mathcal{A}$ are also models of $\mathcal{A}'$. For ABox entailment we write $\mathcal{T} \cup \mathcal{A} \models \mathcal{A}'$.

TBox inference services are provided by the systems CEL [2], Fact++
[127], KAON2 [73], Pellet [113], QuOnto [32], and RacerPro [51]. The latter four systems also support ABox inferences services.

3.2 Retrieval Inference Services

For practical applications another set of inference services deals with finding individuals in ABoxes that satisfy certain conditions.

The *retrieval* inference problem is to find all individuals mentioned in an ABox that are instances of a certain concept C. The set of *fillers* of a role R for an individual i w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ is defined as $\{\mathsf{x} \mid (\mathcal{T}, \mathcal{A}) \models (\mathsf{i}, \mathsf{x}) : \mathsf{R}\}$ where $(\mathcal{T}, \mathcal{A}) \models ax$ means that all models of $\mathcal{T}$ and $\mathcal{A}$ also satisfy ax. The set of *roles* between two individuals i and j w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ is defined as $\{\mathsf{R} \mid (\mathcal{T}, \mathcal{A}) \models (\mathsf{i}, \mathsf{j}) : \mathsf{R}\}$.

In practical systems such as RACER, there are some auxiliary queries supported: retrieval of all concept names or all individuals mentioned in a knowledge base, retrieval of the set of all roles, retrieval of role parents and role children (defined analogously to the concept parents and children, see above), retrieval of the set of individuals in the domain and in the range of a role, etc. As a distinguishing feature to other systems, which is important for many applications, we would like to emphasize that RACER supports multiple TBoxes and ABoxes. Assertions can be added to ABoxes after queries have been answered. In addition, RACER also provides support for retraction of assertions in particular ABoxes.

In addition to the basic infererence service *concept-based instance retrieval*, in practical applications more expressive query languages are required.

A 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 written $\{(X_1, \ldots, X_n) \mid atom_1, \ldots, atom_2\}$.

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), union ($\vee$) and negation ($\neg$) (note that the latter refer to atom negation not concept negation and, for instance, negation as failure semantics is assumed in [129]). In addition, a *projection* operator is supported in order to reduce the dimensionality of an intermediate tuple set. This operator is particularly important in combination with negation (complement). For details see [129].

Answering queries in DL systems goes beyond query answering in relational databases. In databases, query answering amounts to model checking (a database instance is seen as a model of the conceptual schema). Query answering w.r.t. TBoxes and ABoxes must take all models into account, and thus requires deduction. The aim is to define expressive but decidable query languages. Well known classes of queries such as *conjunctive queries* and *unions of conjunctive queries* are topics of current investigations in this context.

In the literature (e.g., [68; 45; 130]), two different semantics for these kinds of queries are discussed. In *standard* conjunctive queries, variables are bound to (possibly anonymous) domain objects. In so-called *grounded* conjunctive queries, variables are bound to named domain objects (object constants). However, in grounded conjunctive queries the standard semantics can be obtained for so-called tree-shaped queries by using existential restrictions in query atoms.

ABox entailment can be reduced to query answering. An ABox $\mathcal{A}'$ is entailed by a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ if for all assertions α in $\mathcal{A}'$ it holds that the boolean query $\{() \mid \alpha\}$ returns *true* (the head is empty for boolean queries since we are not interested in variable bindings).

$$
\begin{aligned}
ImageQuery_1 &:= \{(X, Y) \mid \quad MailDelivery(X), Bicycle(Y), hasPart(X, Y)\} \\
URLQuery_1 &:= \{(X, value(X)) \mid \quad hasURL(mail\_deliv_1, X)\} \\
URLQuery_2 &:= \{(X, value(X)) \mid \quad hasURL(bicycle_1, X)\}
\end{aligned}
$$

Figure 3.1: Query for "a mail delivery with a bicycle" and subsequent queries for retrieving the URLs w.r.t. the result for $ImageQuery_1$.

A query which might be posed in an information system is shown in Figure 3.1. As a result, the inference system returns the tuple $(mail\_deliv_1, bicycle_1)$, and in order to show the image (and high-

light the area with the bicycle), the associated URL names can be retrieved (see also Figure 3.1). The form $value(x)$ returns a unique binding for a variable (in this case a string) if it exists, and $\emptyset$ otherwise. In case of $URLQuery_1$ the answer is $(url_1, \texttt{"http://www.img.de/image-1.jpg"})$. The result of $URLQuery_1$ is defined analogously. The URLs can be used to actually retrieve the image data. Subsequent queries w.r.t. the annotation individuals $mail\_deliv_1$ and $bicycle_1$ are certainly possible. We do not discuss details here, however. In summary, it should be clear now, how annotations with metadata are used in an ontology-based information retrieval system.
a query for a *Postal_Employee* or a *Postman* will also return the media object shown in Figure 1.1. In general, all benefits of description logic reasoning carry over to query answering in an information retrieval system of the kind sketched above.

It is easy to see that annotations such as the one shown in Figure 2.2 can be set up such that the URLs are tied to the ABox individuals comprising the high-level descriptions. In particular, one can easily imagine a situation in which there exist multiple interpretations of an image, which results in multiple annotations being associated with an image. In addition, it is obvious that a repository of media objects together with their annotations (metadata) gives rise to one or more ABoxes that are managed by the ontology-based information system. Not so obvious is how metadata can be automatically derived since manual annotation is too costly in almost all practical scenarios. The derivation of metadata representing high-level interpretation of media content is discussed in the next section.

3.3 Nonstandard Inference Services

In addition to standard inference services and retrieval inference services, recently another set of inference problems have been defined, decision problems have been shown to be decidable, and practical inference algorithms as well as system implementations have been developed (cf., [128]). These services are known as non-standard inference services. Non-standard inference services are useful for building ontologies from examples (learning or bottom-up approach for constructing ontologies). Note that depending on the ontology language, a solution for the problems mentioned below need not necessarily exist. Furthermore, algorithms for the problems are known only for non-expressive description logics in most cases.

- Least-common subsumers (LCS)
  [13; 19; 3]
  A least-common subsumer represents the commonalities between a set of concepts. For a given set of concepts C_1,...,C_n their least-common subsumer E is defined as follows:

  1. $C_i \sqsubseteq E$ for all i = 1,...,n and
  2. If E' is a concept satisfying $C_i \sqsubseteq E'$ for all i = 1,...,n then $E \sqsubseteq E'$.

- Most-specific concept [26]
  Given a finite set of assertions in *ABox* $\mathcal{A}$ of the form $C(a)$ or $r(a, b)$, where C is a concept and r is a role name, the concept E represents the most-specific concept of the individual a in *ABox* $\mathcal{A}$ if it satifies:

  1. $\mathcal{A} \models E(a)$, and
  2. If E' is a concept satisfying $\mathcal{A} \models E'(a)$ then $E \sqsubseteq E'$.

- Rewriting w.r.t. terminologies [20]
  The aim is to rewrite a given concept definition C w.r.t. to a terminology T into an equivalent concept C', such that redundancies are removed and its length is reduced. The resulting concept definition C' is easier to understand by humans than the previous definition C. In an ideal world the objective is a *minimal rewriting*, such that given an ordering $\preceq$ on concept definitions, a rewriting C' is *minimal* iff there does not exist a rewriting C'' such that $C'' \preceq C'$. Obtaining the *minimal* concept rewriting is computationally non-trivial, therefore it is sufficient to compute a *small* but not *minimal* definition.

- Approximation [28; 27]
  Given a concept description written in a more expressive DL, the aim is to translate the concept into a less expressive DL with minimal loss of information. Thus, given two DL languages, e.g. $\mathcal{L}_1 = \mathcal{ALC}$ and $\mathcal{L}_2 = \mathcal{ALE}$ an *approximation* of a concept description C in $\mathcal{L}_1$ to a concept description in $\mathcal{L}_2$ is a concept description D, such that $C \sqsubseteq D$ and D is minimal w.r.t. subsumption.

- Difference [27]
  The difference E of two concepts C and D is defined as follows: $E \sqcap D \equiv C$.

- Matching [18; 14; 15; 16; 17; 4]
  The aim is to locate similar concepts in one terminology in order to find redundancies, or similar concepts in different ontologies to help in their integration. Matching is based on concept patterns. A pattern substitutes the concept terms that appear in a concept description with *concept variables*, such *concept variables* can represent any arbitrary concept description. Thus, matching is based on finding concepts that have similar patterns. This is formally defined as follows: given a concept pattern C and a concept term D, find a substitution σ of the concept variables in C with concept terms such that $\sigma(C) \equiv \mathrm{D}$.

- Explanation (axiom pinpointing) [77; 75]
  To support ontology engineers in finding the causes of errors or inconsistencies in an ontology, axiom pinpointing aims to identify those parts of the ontology causing an unintended ramification (e.g., an ABox inconsistency). In this context, the explanation service should provide arguments that prove why the previously pinpointed axioms hold for the unintended ramification.

- Abduction [40]
  Abduction aims to derive a set of explanations Δ for a given set of assertions Γ such that Δ is consistent w.r.t. to the ontology $(\mathcal{T}, \mathcal{A})$ and satisfies:

  1. $\mathcal{T} \cup \mathcal{A} \cup \Delta \models \Gamma$ and
  2. If Δ' is an ABox satisfying $\mathcal{T} \cup \mathcal{A} \cup \Delta' \models \Gamma$, then $\Delta \models \Delta'$.
  3. $\Gamma \not\models \Delta$.

3.4 Systems

New versions of the description logic systems discussed in the previous section have been developed. These systems are FaCT++ (for $\mathcal{SHOIQ}(\mathcal{D})$) [127] and RacerPro (at the time of this writing the latter only provides an approximation for nominals). FaCT++ is written in C++ whereas RacerPro is implemented in CommonLisp. A new Java-based description logic system for $\mathcal{SHOIQ}(\mathcal{D})$ (and OWL DL) is Pellet. As FaCT++, Pellet is based on a tableau reasoning algorithm and integrates various optimization techniques in order to provide for a fast and efficient practical implementation. New developments also tackle the problem of "repairing" knowledge bases in case an inconsistency is detected [76]. In addition, with Pellet, optimization techniques, for instance, for nominals have been investigated [112]. Other description logic systems are described in [95].

Compared to initial approaches for query languages (see [82]), recently, more expressive languages for instance retrieval have been investigated (conjunctive queries [33; 71; 46]). To the best of the authors' knowledge, algorithms for answering conjunctive queries for expressive description logics such as $\mathcal{SHIQ}$ are not known. In practical systems such as Racer implementations for a restricted form of conjunctive queries is available (variables are bound to individuals mentioned in the ABox only). Database-inspired optimization techniques for this language in the context of a tableau prover are presented in [96]. In addition, Racer supports the incremental computation of result sets for restricted conjunctive queries. The demand for efficient instance retrieval has led to the development of a new proof technique for languages of the $\mathcal{SH}$ family. A transformation approach using disjunctive Datalog [39], resolution techniques as well as magic-set transformations to support reasoning for $\mathcal{SHIQ}$ is described in [73; 97] with encouraging results. In this context,

a new system, KAON2 has demonstrated that techniques from the database community can be successfully used also for implementing description logic systems although, at the time of this writing, KAON2 is a very recent development and not quite as expressive as FACT++, PELLET, or RACERPRO (e.g., w.r.t. datatypes, nominals, large numbers in qualified number restrictions, etc.). A system supporting the fuzzy description logics f-$\mathcal{SHIN}$ [117] is called FIRE.

The synergistic approach of BOEMIE, realized by the integration of different components in an open architecture can profit from the recent advances in the development of standards for description logic reasoning systems (such as DIG [24]). Standards have contributed to the fact that DL systems can be interchanged such that the strength of particular reasoning systems can be exploited for building practical applications. Since semantic web applications have become interesting from a business point of view, commercial DL systems have appeared (e.g., CEREBRASERVER from Cerebra Inc.) and commercial versions of above-mentioned systems became available (e.g., KAON2 from Ontoprise or RACERPRO from Racer-Systems and Franz Inc.).

Bibliography

[1] E. Achilles, B. Hollunder, A. Laux, and J. P. Mohren. $\mathcal{KRIS}$: Knowledge Representation and Inference System – User guide. Technical Report D91-14, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1991.

[2] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

[3] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 400–412, Lisbon, Portugal, 2004. Springer-Verlag.

[4] Franz Baader, Sebastian Brandt, and Ralf Küsters. Matching under side conditions in description logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 213–218, 2001.

[5] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality Restrictions on Concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[6] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[7] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[8] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[9] Franz Baader and Philipp Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German Workshop on Artificial Intelligence (GWAI'92)*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 1992.

[10] Franz Baader and Bernhard Hollunder. $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. *SIGART Bull.*, 2(3):8–14, 1991.

[11] Franz Baader and Bernhard Hollunder. Cardinality restrictions on concepts. Technical Report RR-03-48, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1993.

[12] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 270–281. Morgan Kaufmann, 1992.

[13] Franz Baader and Ralf Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proc. of the 22nd German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1998.

[14] Franz Baader and Ralf Küsters. Matching in description logics with existential restrictions. In *Proc. of the 1999 Description Logic Workshop (DL'99)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-22/`, 1999.

[15] Franz Baader and Ralf Küsters. Matching in description logics with existential restrictions — Revisited. Technical Report LTCS-Report 99-13, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[16] Franz Baader and Ralf Küsters. Matching in description logics with existential restrictions. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 261–272, 2000.

[17] Franz Baader and Ralf Küsters. Unification in a description logic with transitive closure of roles. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proc. of the 8th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, volume 2250 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2001.

[18] Franz Baader, Ralf Küsters, Alex Borgida, and Deborah L. McGuinness. Matching in description logics. *J. of Logic and Computation*, 9(3):411–447, 1999.

[19] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–101, 1999.

[20] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting concepts using terminologies. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 297–308, 2000.

[21] Franz Baader, Ralf Küsters, and Frank Wolter. Extensions to description logics. In Baader et al. [6], chapter 6, pages 219–261.

[22] Franz Baader and Werner Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[23] F. Bacchus. *Representing and reasoning with probabilistic knowledge: A logical approach to probabilities*. The MIT Press, Cambridge, 1990.

[24] Sean Bechhofer, Volker Haarslev, Ralf Möller, and Peter Crowthe. The dig description logic interface. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, 2003.

[25] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagram using description logic based system. In *Proc. of the KI-2001 Workshop W6 on Applications of Description Logics ADL-01*, volume 44. http://ceur-ws.org/Vol-44/, 2001.

[26] Alexander Borgida and Ralf Küsters. What's not in a name? Initial explorations of a structural approach to integrating large concept knowledge-bases. Technical Report DCS-TR-391, Rutgers University, 1999.

[27] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 203–214, San Francisco, CA, 2002. Morgan Kaufman.

[28] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation in description logics. LTCS-Report 01-06, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2001. Available at `http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html`.

[29] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: Preliminary report. In *Proc. of the 1995 Description Logic Workshop (DL'95)*, pages 131–139, 1995.

[30] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI'93)*, pages 704–709, 1993.

[31] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[32] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[33] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[34] Z. Ding and Y. Peng. A probabilistic extension to ontology language OWL. In *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS)*, 2004.

[35] Z. Ding, Y. Peng, and R. Pan. BayesOWL: Uncertainty modeling in semantic web ontologies. In *Soft Computing in Ontologies and Semantic Web*. Springer, 2005.

[36] Francesco M. Donini. Complexity of reasoning. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 96–136. Cambridge University Press, 2003.

[37] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 151–162, 1991.

[38] M. Dürig and Th. Studer. Probabilistic abox reasoning: Preliminary results. In *Proc. Int. Description Logics Workshop 2005*, pages 104–111, 2005.

[39] Thomas Eiter, Georg Gottlob, and Heikki Mannilla. Disjunctive Datalog. *ACM Trans. on Database Systems*, 22(3):364–418, 1997.

[40] C. Elsenbroich, O. Kutz, and U. Sattler. A case for abductive reasoning over ontologies. In *Proc. OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11*, 2006.

[41] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

[42] N. Fuhr. Probabilistic Datalog: a logic for powerful retrieval methods. In *Proceedings of SIGIR-95: 18th ACM International Conference on Research and Development in Information Retrieval*, pages 282–290, 1995.

[43] N. Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society of Information Science*, 51(2):95–110, 2000.

[44] R. Giugno and T. Lukasiewicz. P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 86–97. Springer-Verlag, 2002.

[45] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*. AAAI Press, 2007.

[46] Birte Glimm and Ian Horrocks. Handling cyclic conjunctive queries. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, volume 147. CEUR (`http://ceur-ws.org/`), 2005.

[47] Volker Haarslev and Ralf Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 273–284, 2000.

[48] Volker Haarslev and Ralf Möller. Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 152–161. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-49/`, 2001.

[49] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[50] Volker Haarslev and Ralf Möller. Optimizing reasoning in description logics with qualified number restrictions. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 142–151. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-49/`, 2001.

[51] Volker Haarslev and Ralf Möller. RACER System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001. Available at `http://www.racer-systems.com`.

[52] Volker Haarslev and Ralf Möller. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004.

[53] Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.

[54] Volker Haarslev, Ralf Möller, and Michael Wessel. The description logic $\mathcal{ALCNH}_{R+}$ extended with concrete domains: A practically motivated approach. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, pages 29–44, 2001.

[55] J.Y. Halpern. An analysis fo first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[56] Jochen Heinsohn. Probabilistic description logics. In Ramon Lopez de Mantaras and David Poole, editors, *Proc. of the 10th Conf. on Uncertainty in Artificial Intelligence*, pages 311–318, Seattle, Washington, 1994. Morgan Kaufmann.

[57] Berhnard Hollunder, Armin Laux, Hans-Jürgen Profitlich, and Th. Trenz. $\mathcal{KRIS}$-manual. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1991.

[58] Bernhard Hollunder. *Algorithmic Foundations of Terminological Knowledge Representation Systems*. PhD thesis, University of Saarbrücken, Department of Computer Science, 1994.

[59] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, 1991.

[60] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. Technical Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1990.

[61] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[62] Ian Horrocks. The FaCT system. In Harrie de Swart, editor, *Proc. of the 7th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 307–312. Springer, 1998.

[63] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[64] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

[65] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[66] Ian Horrocks and Ulrike Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281, July 2002.

[67] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[68] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stefan Tobies. How to decide query containment under constraints using a description logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[69] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[70] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[71] Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[72] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[73] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing $\mathcal{SHIQ}$-Description Logic to Disjunctive Datalog Programs. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, 2004.

[74] Manfred Jaeger. Probabilistic reasoning in terminological logics. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 305–316, 1994.

[75] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca-Grau. Repairing unsatisfiable concepts in owl ontologies. In *Proc. The European Semantic Web Conference (ESWC)*, 2006.

[76] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in owl ontologies. *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*, 3(4), 2005.

[77] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in owl ontologies. *Journal of Web Semantics*, 3(4), 2006.

[78] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI), Madison, Wisconsin*, 1998.

[79] Daphne Koller, Alon Levy, and Avi Pfeffer. P-CLASSIC: A tractable probabilistic description logic. In *Proc. of the 14th Nat. Conf. on Artificial Intelligence (AAAI'97)*, pages 390–397. AAAI Press/The MIT Press, 1997.

[80] Patrick Lambrix. *Part-Whole Reasoning in Description Logic*. PhD thesis, Dep. of Computer and Information Science, Linköping University, Sweden, 1996.

[81] Daniel J. Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15(1):61–82, 1995.

[82] Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence (AAAI'91)*, pages 471–476, 1991.

[83] Hongkai Liu, Carsten Lutz, Maya Milicic, and Frank Wolter. Description logic actions with general TBoxes: a pragmatic approach. In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, 2006.

[84] Th. Lukasiewicz. Probabilistic description logic programs. In *Proc. of ECSQARU*, pages 737–749, 2005.

[85] Th. Lukasiewicz. Stratified probabilistic description logic programs. In *Proc. of ISWC-URSW*, pages 87–97, 2005.

[86] Carsten Lutz. The complexity of reasoning with concrete domains. LTCS-Report 99-01, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[87] Carsten Lutz. Complexity of terminological reasoning revisited. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 181–200. Springer, 1999.

[88] Carsten Lutz. Reasoning with concrete domains. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 90–95, 1999.

[89] Carsten Lutz. Interval-based temporal reasoning with general TBoxes. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 89–94, 2001.

[90] Carsten Lutz. NEXPTIME-complete description logics with concrete domains. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 45–60. Springer, 2001.

[91] Carsten Lutz. Description logics with concrete domains: A survey. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logics*, volume 4. King's College Publications, 2003.

[92] Carsten Lutz. Nexptime-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

[93] Carsten Lutz and Maja Milicic. A tableau algorithm for description logics with concrete domains and gcis. In *Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods TABLEAUX 2005*, LNAI, Koblenz, Germany, 2005. Springer.

[94] Ralf Möller. Expressive description logics: Foundations for applications, 2000. Habilitation Thesis.

[95] Ralf Möller and Volker Haarslev. Description logic systems. In Baader et al. [6], chapter 8, pages 282–305.

[96] Ralf Möller, Volker Haarslev, and Michael Wessel. On the scalability of description logic instance retrieval. In Chr. Freksa, Kohlhase, and K. M. Schill, editors, *29. Deutsche Jahrestagung für Künstliche Intelligenz*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2006. forthcoming.

[97] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univ. Karlsruhe, 2006.

[98] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. of Web Semantics*, 3(1):41–60, 2005.

[99] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[100] N. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[101] H. Nottelmann and N. Fuhr. Learning probabilistic datalog rules for information classification and transformation. In *In Proceedings CIKM*, pages 387–394, 2001.

[102] H. Nottelmann and N. Fuhr. pDAML+OIL: A probabilistic extension to DAML+OIL based on probabilistic datalog. In *Proceedings Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2004.

[103] H. Nottelmann and N. Fuhr. Adding probabilities and rules to OWL Lite subsets based on probabilistic datalog. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 14(1):17–41, 2006.

[104] Hans Jurgen Ohlbach and Jana Koehler. Modal logics, description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1-2):1–31, 1999.

[105] Jeff Z. Pan, Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras, and Ian Horrocks. f-swrl: A fuzzy extension of swrl. *Data Semantics, special issue on Emergent Semantics*, 4090:28–46, 2006.

[106] A. Pfeffer, D. Koller, B. Milch, and K.T. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proc. of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 541–550, 1999.

[107] D. Poole. First-order probabilistic inference. In *Proc. International Joint Conference on Artificial Intelligence IJCAI-03*, pages 985–991, 2003.

[108] Ulrike Sattler. A concept language extended with different kinds of transitive roles. In Günter Görz and Steffen Hölldobler, editors, *Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96)*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer, 1996.

[109] Ulrike Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 1998.

[110] Andrea Schaerf. Reasoning with individuals in concept languages. In *Proc. of the 3rd Conf. of the Ital. Assoc. for Artificial Intelligence (AI*IA'93)*, Lecture Notes in Artificial Intelligence. Springer, 1993.

[111] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[112] Evren Sirin, Bernardo Cuenca Grau, and Bijan Parsia. From wine to water: Optimizing description logic reasoning for nominals. In *International Conference on the Principles of Knowledge Representation and Reasoning (KR-2006)*, 2006.

[113] Evren Sirin and Bijan Parsia. Pellet System Description. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[114] C. Smyth and D. Poole. Qualitative probabilistic matching with hierarchical descriptions. In *Proc. Knowledge Representation and Reasoning (KR&R 2004)*, 2004.

[115] P.-H. Speel, F. van Raalte, P. E. van der Vet, and N. J. I. Mars. Runtime and memory usage performance of description logics. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium*, pages 13–27, 1995.

[116] G. Stoilos, G. Stamou, and J.Z. Pan. Handling imprecise knowledge with fuzzy description logic. In *International Workshop on Description Logics (DL 06), Lake District, UK*, 2006.

[117] G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and I. Horrocks. The fuzzy description logic f-shin. In *International Workshop on Uncertainty Reasoning For the Semantic Web*, 2005.

[118] G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and I. Horrocks. A fuzzy description logic for multimedia knowledge representation. In *Proc. of the International Workshop on Multimedia and the Semantic Web*, 2005.

[119] G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and I. Horrocks. Fuzzy owl: Uncertainty and the semantic web. In *International Workshop of OWL: Experiences and Directions, Galway*, 2005.

[120] G. Stoilos, U. Straccia, G. Stamou, and Jeff Z. Pan. General concept inclusions in fuzzy description logics. In *17th European Conference on Artificial Intelligence (ECAI 06), Riva del Garda, Italy*, 2006.

[121] Umberto Straccia. Reasoning within fuzzy description logics. *J. of Artificial Intelligence Research*, 14:137–166, 2001.

[122] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[123] Christopher B. Tresp and Ralf Molitor. A description logic for vague knowledge. In *Proc. of the 13th Eur. Conf. on Artificial Intelligence (ECAI'98)*, pages 361–365, 1998.

[124] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, volume 104. CEUR (`http://ceur-ws.org/`), 2004.

[125] Dmitry Tsarkov and Ian Horrocks. Optimised classification for taxonomic knowledge bases. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, volume 147. CEUR (`http://ceur-ws.org/`), 2005.

[126] Dmitry Tsarkov and Ian Horrocks. Ordering heuristics for description logic reasoning. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 609–614, 2005.

[127] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, 2006. To appear.

[128] Anni-Yasmin Turhan and Christian Kissig. SONIC—Non-standard inferences go OILED. In *Proc. of the 2nd Int. Joint Conf. on Automated Reasoning (IJCAR 2004)*, volume 3097 of *Lecture Notes in Computer Science*, pages 321–325. Springer, 2004.

[129] M. Wessel and R. Möller. A High Performance Semantic Web Query Answering Engine. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proc. International Workshop on Description Logics*, 2005.

[130] M. Wessel and R. Möller. A flexible dl-based architecture for deductive information systems. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proc. IJCAR-06 Workshop on Empirically Successful Computerized Reasoning (ESCoR)*, pages 92–111, 2006.

[131] Philip Y. Yelland. An alternative combination of Baysian networks and description logics. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 225–234, 2000.

[132] John Yen. Generalizing term subsumption languages to fuzzy logic. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 472–477, 1991.

Part II

Instance Retrieval: Optimization Techniques for Scalable Reasoning

Chapter 4

Optimization Techniques for Instance Retrieval

Although description logics (DLs) become more and more expressive (e.g., [10]), due to our experiences only for some tasks the expressivity of description logics really comes into play whereas for many applications, it is also necessary to be able to deal with largely deterministic knowledge very effectively. Thus, in practice, description logic systems offering high expressivity must also be able to handle large bulks of data descriptions (ABoxes) which are largely deterministic. Users expect that DL systems scale w.r.t. these practical needs. In our view there are two kinds of scalability problems: scalability w.r.t. large sets of data descriptions (data description scalability) and scalability w.r.t. high expressivity, which might only be important for small parts of the data descriptions (expressivity scalability).

In the literature, the data description scalability problem has been tackled from different perspectives. We see two main approaches, the layered approach and the integrated approach. In the layered approach the goal is to use databases for storing and accessing data, and exploit description logic ontologies for convenient query formulation. The main idea is to support ontology-based query translation to relational query languages (SQL, Datalog). See, e.g., [12, 6] (DLDB), [2] (Instance Store), or [3] (DL-Lite). We notice that these approaches are only applicable if reduced expressivity does not matter. Despite the most appealing argument of reusing database technology (in particular services for persistent data), at the current state of the art it is not clear how expressivity can be increased to, e.g., $\mathcal{SHIQ}$ without losing the applicability of transformation approaches. Hence, while data description scalability is achieved, it is not clear how to extend these approaches to achieve expressivity scalability (at least for some parts of the data descriptions).

Tableau-based DL systems are now widely used in practical applications because these systems are quite successful w.r.t. the expressivity scalability problem. Therefore, for investigating solutions to both problems, the expressivity and the data description scalability problem, we pursue the integrated approach that considers query answering with a tableau-based description logic system augmented with new techniques inspired from database systems. For the time being we ignore the persistency problem and investigate specific knowledge bases (see below).

We present and analyze the main results we have obtained on how to start solving the scalability problem with tableau-based prover systems given large sets of data descriptions for a large number of individuals. Note that we do not discuss query answering speed of a particular system 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. Since DLs are very popular now, and tableau-based systems have been extensively studied in the literature (see [1] for references), we assume the reader is familiar tableau-based decision procedures.

In order to investigate the data description scalability problem, we use the Lehigh University BenchMark (LUBM, [5, 6]). LUBM queries are conjunctive queries referencing concept, role, and individual names from the TBox. A query language tailored to description logic applications that can express these queries is described in [11] (the language is called nRQL). Variables are bound

to individuals mentioned in the ABox.[1]

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 [5, 6] for more information about the LUBM queries.

$$Q9 : ans(x, y, z) \leftarrow Student(x), Faculty(y), Course(z), \\ advisor(x, y), takesCourse(x, z), teacherOf(y, z)$$

$$Q12 : ans(x, y) \leftarrow Chair(x), Department(y), memberOf(x, y), \\ subOrganizationOf(y,' www.University0.edu')$$

In order to investigate the data description scalability problem, we used a TBox for LUBM with 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, there is an axiom $Chair \doteq Person \sqcap \exists headOf.Department$. In the evaluation of optimization techniques for query answering, we consider runtimes for a whole query set (queries 1 to 14 in the LUBM case).

If the queries mentioned above are answered in a naive way by evaluating subqueries in the sequence of syntactic notation, acceptable answering times can hardly be achieved. Determining all bindings for a variable (with a so-called generator, see below) is much more costly than verifying a particular binding (with a tester). Treating the one-place predicates $Student$, $Faculty$, and $Course$ as generators of bindings for corresponding variables results in combinatorial explosion (cross product computation). Optimization techniques are required that provide for efficient query answering in the average case.

4.1 Query Optimization

The optimization techniques that we investigated are inspired by database join optimizations, and exploit the fact that there are few $Faculties$ but many $Students$ in the data descriptions. Predicates used in queries can be used as *generators* and as *tester*. Generators are used to compute bindings of variables (i.e., individuals that "satisfiy" the predicates) whereas testers (also called filters) are used to test whether given bindings (i.e., individuals) satisfy the test predicates. A binary atom can additionally also take the role of a so-called *successor generator* (or *predecessor generator*) if only the variable on the righthand side (or lefthand side) has no binding at the moment in which the atom is evaluated.

Testers are much more efficient that generators, and query answering times depend on the selection of generators that produce as few bindings as possible. For instance, in case of query Q9 from LUBM, the idea is to use $Faculty$ as a generator of bindings for y and then generate the bindings for z following the role $teacherOf$. The heuristic applied here is that the average cardinality of a set of role fillers is rather small. For the given z bindings we apply the predicate $Course$ as a tester (rather than as a generator as in the naive approach). Given the remaining bindings for z, bindings for x can be established via the inverse of $takesCourse$. These x bindings are then filtered with the tester $Student$.

If z was not mentioned in the set of variables for which bindings are to be computed, and the tester $Course$ was not used, there would be no need to generate bindings for z at all. One could just check for the existence of a $takesCourse$ role filler for bindings w.r.t. x.

In the second example, query Q12, the constant $'www.University0.edu'$ is mentioned. Starting from this individual the inverse of $subOrganizationOf$ is applied as a generator for bindings for y which are filtered with the tester $Department$. With the inverse of $memberOf$, bindings for x are computed which are then filtered with $Chair$. Since for the concept $Chair$ sufficient conditions are declared in the TBox, instance retrieval reasoning is required if $Chair$ is a generator. Thus, it is advantageous that $Chair$ is applied as a tester (and only instance tests are performed).

For efficiently answering queries, a query execution plan is determined by a cost-based optimization component (c.f., [4, p. 787ff.]), which orders query atoms in such a way that queries can be answered effectively. For computing a total order relation on query atoms with respect to a given set of data descriptions (assertions in an ABox), we need information about the number

[1] In the notation of conjunctive queries used in this paper we assume that different variables may have the same bindings.

of instances of concept and role names. An estimate for this information can be computed in a preprocessing step by considering given data descriptions, or could be obtained by examining the result set of previously answered queries (we assume that ABox realization is too costly, so this alternative is ruled out).

4.2 Indexing by Exploiting Told and Taxonomical Information

In many practical applications that we encountered, data descriptions often directly indicate of which concept an individual is an instance. Therefore, in a preprocessing step, it is useful to compute an index that maps concept names to sets of individuals which are their instances. In a practical implementation this index will be realized with some form of hash table.

Classifying the TBox yields the set of ancestors for each concept name, and if an individual i is an instance of a concept name A due to explicit data descriptions, it is also an instance of the ancestors of A. The index is organized in such a way that retrieving the instances of a concept A, or one of its ancestors, requires (almost) constant time. The index is particularly useful for providing bindings of variables if, despite all optimization attemps for deriving query execution plans, concept names must be used as generators. In addition, the index is used to estimate the cardinality of concept extensions. The estimates are used to compute an order relation for query atoms. The smaller the cardinality of a concept or a set of role fillers is assumed to be, the more priority is given to the query atom. Optimizing LUBM query $Q9$ with the techniques discussed above yields the following query execution plan.

$$Q9' : ans(x,y,z) \leftarrow Faculty(y), teacherOf(y,z), Course(z), \\ advisor^{-1}(y,x), Student(x), takesCourse(x,z)$$

Using this kind of rewriting, queries can be answered much more efficiently.

If the TBox contains only GCIs of the form $A \sqsubseteq A_1 \sqcap \ldots \sqcap A_n$, i.e., if the TBox forms a hierachy, the index-based retrieval discussed in this section is complete (see [2]). However, this is not the case for LUBM. In LUBM, besides domain and range restrictions, axioms are also of the form $A \doteq A_1 \sqcap A_2 \sqcap \ldots \sqcap A_k \sqcap \exists R_1.B_1 \sqcap \ldots \sqcap \exists R_m.B_m$ (actually, $m = 1$). If sufficient conditions with exists restrictions are specified, as in the case of $Chair$, optimization is much more complex. In LUBM data descriptions, no individual is explicitly declared as a $Chair$ and, therefore, reasoning is required, which is known to be rather costly. If $Chair$ is used as a generator and not as a tester such as in the simple query $ans(x) \leftarrow Chair(x)$, optimization is even more important. The idea to optimize instance retrieval is to detect an additional number of obvious instances using further incomplete tests, and, in addition, to determine obvious non-instances. We first present the latter technique and continue with the former afterwards.

4.3 Obvious Non-Instances: Exploiting Information from one Completion

The detection of "obvious" non-instances of a given concept C can be implemented using a model merging operator defined for so-called individual pseudo models (aka pmodels) as defined in [7]. Since these techniques have already been published, we just sketch the main idea here for the sake of completeness. The central idea is to compute a pmodel from a completion that is derived by the tableau prover.

For instance, in the DL $\mathcal{ALC}$ a pseudo model for an individual i mentioned in a consistent initial A-box A w.r.t. a TBox T is defined as follows. Since A is consistent, there exists a set of completions $\mathcal{C}$ of A. Let $A' \in \mathcal{C}$. An *individual pseudo model* M for an individual i in A is defined as the tuple $\langle M^D, M^{\neg D}, M^{\exists}, M^{\forall} \rangle$ w.r.t. A' and A using the following definition.

$$M^D = \{D \,|\, i : D \in A',\ D \text{ is a concept name}\}$$
$$M^{\neg D} = \{D \,|\, i : \neg D \in A',\ D \text{ is a concept name}\}$$
$$M^{\exists} = \{R \,|\, i : \exists R.C \in A'\} \cup \{R \,|\, (i,j) : R \in A\}$$
$$M^{\forall} = \{R \,|\, i : \forall R.C \in A'\}$$

Note the distinction between the initial A-box A and its completion A'. It is important that all restrictions for a certain individual are "reflected" in the pmodel. The idea of model merging is that there is a simple sound but incomplete test for showing that adding the assertion $i : \neg C$ to the ABox will not lead to a clash (see [7] for details) and, hence, i is not an instance of the query concept C. The pmodel merging test is: $atoms\_mergable(MS) \wedge roles\_mergable(MS)$ where $atoms\_mergable$ tests for a possible primitive clash between pairs of pseudo models. It is applied to a set of pseudo models MS and returns *false* if there exists a pair $\{M_1, M_2\} \subseteq MS$ with $(M_1^D \cap M_2^{\neg D}) \neq \emptyset$ or $(M_1^{\neg D} \cap M_2^D) \neq \emptyset$. Otherwise it returns *true*.

The algorithm $roles\_mergable$ tests for a possible role interaction between pairs of pseudo models. It is applied to a set of pseudo models MS and returns *false* if there exists a pair $\{M_1, M_2\} \subseteq MS$ with $(M_1^{\exists} \cap M_2^{\forall}) \neq \emptyset$ or $(M_1^{\forall} \cap M_2^{\exists}) \neq \emptyset$. Otherwise it returns *true*. The reader is referred to [8] for the proof of the soundness of this technique and for further details.

It should be emphasized that the complete set of data structures for a particular completion is not maintained by a DL reasoner. The pmodels provide for an appropriate excerpt of a completion needed to determine non-instances.

4.4 Obvious Instances: Exploiting Information from the Precompletion

Another central optimization technique to ensure data description scalability as it is required for LUBM is to find "obvious" instances with minimum effort. Given an initial ABox consistency test and a completion, one can consider all deterministic restrictions, i.e., one considers only those completion data structures (from now on called constraints) for which there are no choice points in the tableau proof (in other words, consider only those constraints that do not have dependency information attached). These constraints constitute a so-called precompletion.[2] Note that in a precompletion, no restrictions are violated because we assume that the precompletion is computed from an existing completion.

Given the precompletion constraints, an approximation of the most-specific concept (MSC) of an individual i is computed as follows (the approximation is called MSC'). For all constraints representing role assertions of the form $(i,j) : R$ (or $(j,i) : R$) add constraints of the form $i : \exists R.\top$ (or $i : \exists R^{-1}.\top$). Afterwards, constraints for a certain individual i are collected into a set $\{i : C_1, \ldots, i : C_n\}$. Then, $MSC'(i) := C_1 \sqcap \ldots \sqcap C_n$. Now, if $MSC'(i)$ is subsumed by the query concept C, then i must be an instance of C. In the case of LUBM many of the assertions lead to deterministic constraints in the tableau proof which, in turn, results in the fact that for many instances of a query concept C (e.g., $Faculty$ as in query $Q9$) the instance problem is decided with a subsumption test based on the MSC' of each individual. Subsumption tests are known to be fast due to caching and model merging [9]. The more precisely $MSC'(i)$ approximates $MSC(i)$, the more often an individual can be determined to be an obvious instance of a query concept. It might be possible to determine obvious instances by directly considering the precompletion data structures. However, at this implementation level a presentation would be too detailed. The main point is that, due to our findings, the crude approximation with MSC' suffices to solve many instance tests in LUBM.

If query atoms are used as testers, in LUBM it is the case that in a large number of cases the test for obvious non-instances or the test for obvious instances determines the result. However,

[2] Cardinality measures for concept names, required for determining optimized query execution plans, could be made more precise if cardinality information was computed by considering a precompletion. However, in the case of LUBM this did not result in better query execution plans.

for some individuals i and query concepts C both tests do not determine whether i is an instance of C (e.g., this is the case for $Chair$). Since both of these "cheap" tests are incomplete, for some individuals i a refutational ABox consistency test resulting from adding the claim $i : \neg C$ must be decided with a sound and complete tableau prover. For some concepts C, the set of candidates is quite large. Considering the volume of assertions in LUBM (see below for details), it is easy to see that the refutational ABox consistency test should not start from the initial, unprocessed ABox in order to ensure scalability.

For large ABoxes and many repetitive instance tests it is a waste of resources to "expand" the very same initial constraints over and over again. Therefore, the precompletion resulting from the initial ABox consistency test is used as a starting point for refutational instance tests. The tableau prover keeps the precompletion in memory. All deterministic constraints are expanded, so if some constraint is added, only a limited amount of work is to be done. To understand the impact of refutation-based instance tests on the data description scalability problem, a more low-level analysis on tableau prover architectures is required.

4.5 Index Structures for Optimizing Tableau Provers

Tableau provers are fast w.r.t. backtracking, blocking, caching and the like. But not fast enough if applied in a naive way. If a constraint $i : \neg C$ is added to a precompletion, the tableau prover must be able to very effectively determine related constraints for i that already have been processed. Rather than using linear search through lists of constraints, index structures are required for bulk data descriptions.

First of all, it is relatively easy to classify various types of constraints (for exists restrictions, value restrictions, atomic restrictions, negated atomic restrictions, etc.) and access them effectively according to their type. We call the corresponding data structure an active record of constraint sets (one set for each kind of constraint). For implementing a tableau prover, the issue of an appropriate data structure for these sets arises. Since ABoxes are not models, (dependency-directed) backtracking cannot be avoided in general. In this case, indexing the set of "relevant" constraints in order to provide algorithms for checking if an item is an element of a set or list (element problem) is all but easy. Indexing requires hash tables (or trees), but backtracking requires either frequent copying of index structures (i.e., hash tables) or frequent insertion and deletion operations concerning hash tables. Both operations are known to be costly.

Practical experiments on LUBM with the DL system RacerPro (see below for a detailed evaluation) indicate that the following approach is advantageous in the average case. For frequent updates of the search space structures during a tableau proof, we found that simple lists for different kinds of constraints are most efficient. Thus, we have an active record of lists of constraints. New constraints are added to the head of the corresponding list, which is a very fast operation. During backtracking, the head is chopped off with minimum effort. The list representation is used if there are few constraints, and the element problem (Is an element in a list?) can be decided efficiently. However, if these lists of constraints get large, performance decreases due to linear search. Therefore, if some list from the active record of constraints gets longer than a certain threshold, the record is restructured and the list elements are entered into an appropriate index structure (hash tables with individuals as keys). Then, the tableau prover continues with a new record of empty lists as the active record. The record of lists and associated the hash table are called a generation. New constraints are added to the new active record of constraints and the list(s) of the first generation are no longer used. For the element problem the lists in the active record are examined first (linear search over small lists) and then, in addition, the hash table from the first generation is searched (almost linear search). If a list from the active record gets too large again, a new generation is created. Thus, in general we have a sequence of such generations, which are then considered for the element test in the obvious way. If backtracking occurs, the lists of the appropriate generation are installed again as the active record of lists. This way of dealing with the current search state allows for a functional implementation style of the tableau prover which we prefer for debugging purposes. However, one might also use a destructive way to manage constraints during backtracking. Obviously, all (deterministic) constraints from the initial ABox can be stored in a hash table. In any case, the main point here is that tableau provers need

an individual-based index to efficiently find all constraints in which an individual is involved. In the evaluation of other optimization techniques (see below) we presuppose that a tableau prover is equipped with this technology, and thus, we can assume that each refutational instance test is rather fast.

However, the index structures are copied before new elements are entered because the procedure is subject to backtracking (backjumping to be more precise). If backjumping occurs, the original state is restored (with lists and index structures untouched). Due to space restrictions we cannot go into detail here.

Therefore, if a list of generation n contains a number of elements above a given threshold, generation n is restructured and represented as a hash table. The reference to the original list is retained, and the generation counter n is increased by 1. New constraints are then added to this new generation with empty lists.

In general, there exists a sequence of generations. The *nth* generation is encoded as lists (for frequent addition and removal of constraints) whereas the other generations use hash tables as index structures for efficiently deciding the element problem. In order to solve the element problem, in the worst case all generations are scanned. If search continues in older generations, for each generation the element problem can be solved in almost linear time. The deterministic constraints of the precompletion are stored in a hash table as generation 0. Generation 0 is reused for subsequent ABox consistency tests with some constraints added.

While generation 0 is never subject to backtracking, it may be the case that in a certain branch of the tableau search space, many constraints are generated, causing a new generation m to be created. If backtracking occurs, it might be the case that constraints in generation m must be retracted. If this is the case, the hash table is no longer used and the retained list representation is put into operation again (the hash table might be reused later on using a pool resource). Note that tableau provers do not use chronological backtracking but, usually, some sort of dependency-directed backtracking, namely backjumping. Thus, almost always we jump to a state which corresponds to a list whose length is substantially smaller than the generational threshold for hash table restructuring (see above).

Organizing search is always a tradeoff. Lists are lightweight data structures, i.e., addition of elements and release of list elements during backtracking causes only a small amount of overhead. Hash tables for indexing must be considered as heavyweight data structures, addition and removal are costly operations. Addition is costly because it might involve rehashing as well as collision avoidance operations, deletion of elements in large hash tables causes unnecessary resources to be blocked (rehashing might be possible but would be expensive in terms of time resources). The generational approach has proved to be a good compromise, and corresponding techniques can be employed in different architectures. The index structures considerably speed up the initial ABox consistency test. We now turn back to query answering.

4.6 Evaluation of Implemented Techniques

The significance of the optimization techniques introduced in this contribution is analyzed with the system RacerPro 1.9. RacerPro is freely available for research and educational purposes (`http://www.racer-systems.com`). All experiments to support the claims of this chapter can be verified with the benchmarks found at `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`. The runtimes we present in this section are used to demonstrate the order of magnitude of time resources that are required for solving inference problems. They allow us to analyze the impact of the proposed optimization techniques.

An overview of the size of the LUBM benchmarks is given in Figure 4.1. With an increasing number of universities, there is a linearly increasing number of instances as well as concept and role assertions. For instance, with 50 universities, 1.000.000 instances have to be handled.

The runtimes for answering all 14 LUBM queries are presented in Figures 4.2 and 4.3 (Sunfire, Solaris, 32 GB). In Figure 4.2 a version of the LUBM TBox is used that does not cause backtracking during ABox satisfiability (or consistency) tests . With this kind of benchmark, we can evaluate storage management and indexing techniques of DL provers. In Figure 4.3 we used a variant of the TBox that causes backtracking.

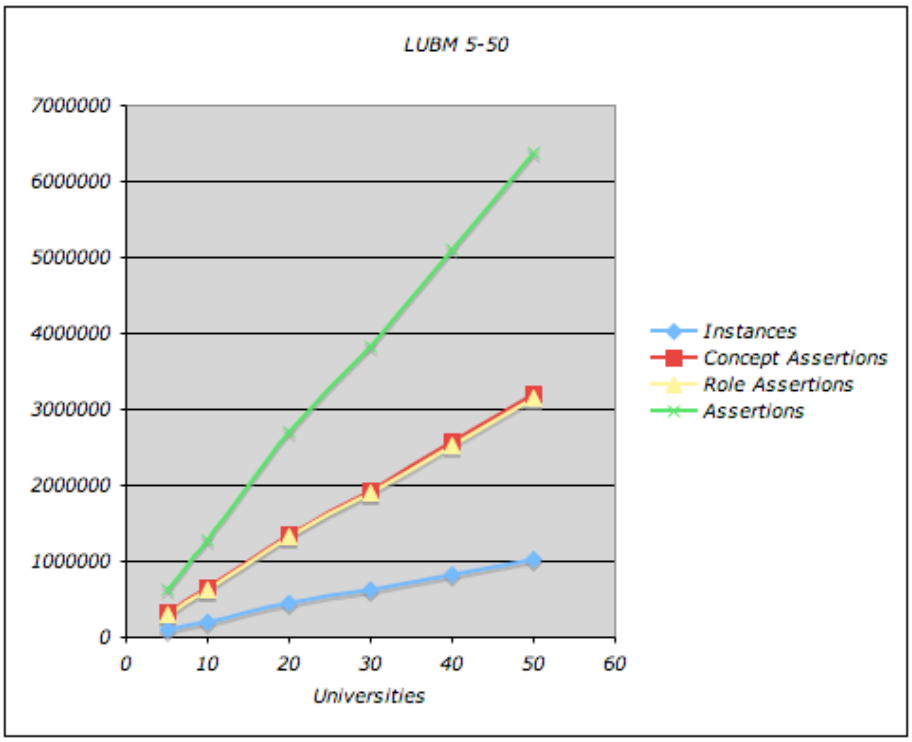

Figure 4.1: Linearly increasing number of individuals, concept assertions and role assertions for different numbers of universities.

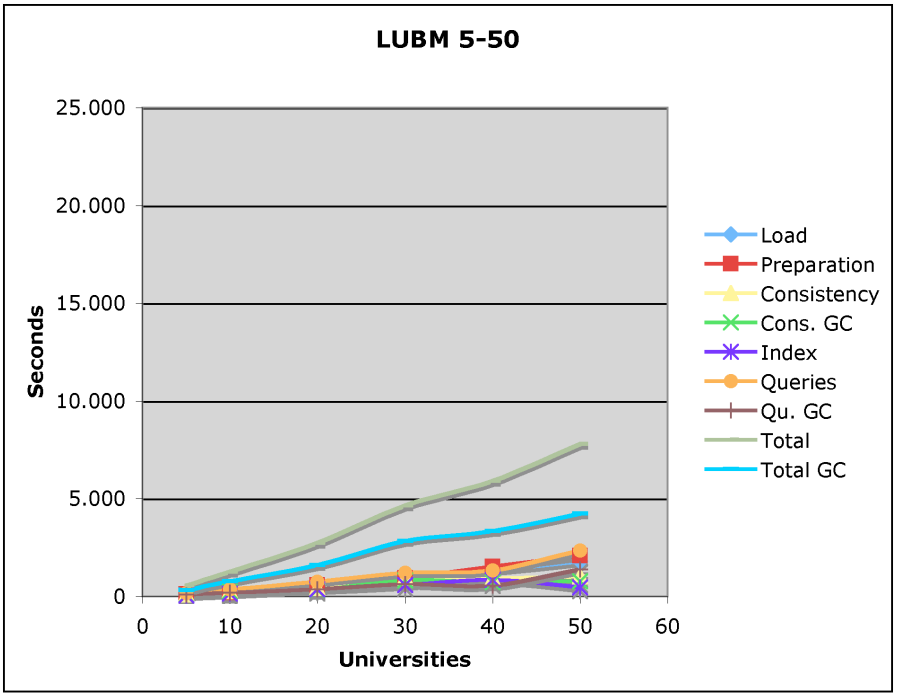

Figure 4.2: Runtimes for deterministic version of LUBM

The time for loading the OWL files is indicated with the curve named “Load”. Mapping syntactic structures from the OWL file to internal structures for compuational processes is called

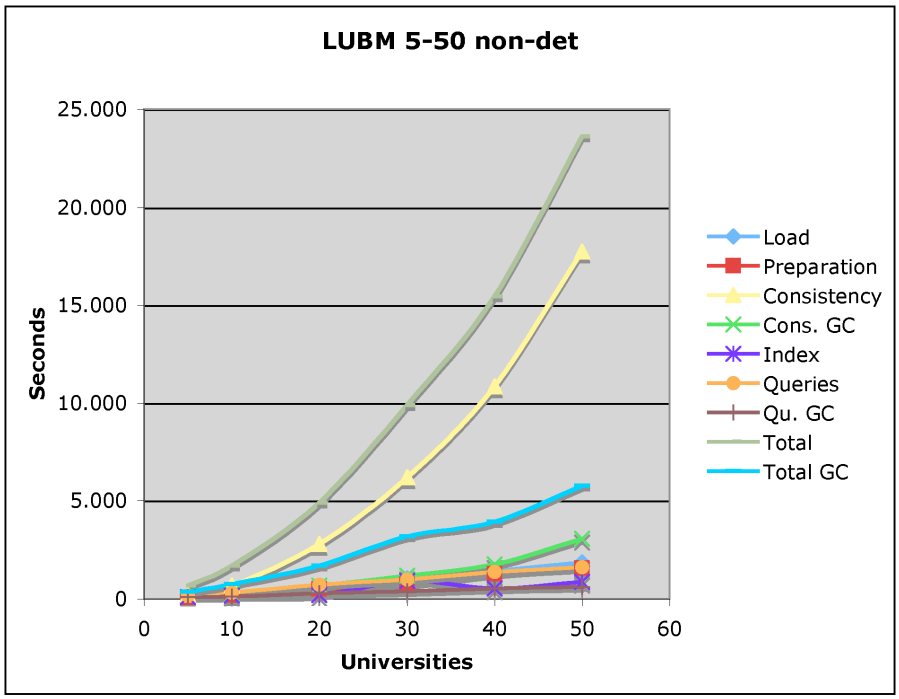

Figure 4.3: Runtimes for non-deterministic version of LUBM

"Preparation". Times for building index structures for retrieval is indicated with the "Index" curve. As can be seen in Figures 4.2 and 4.3 loading, preparation, and indexing can be neglected. Before queries can be answered, the ABox is checked for consistency (see the "Consistency" curve). As can be expected, if there is no backtracking state-of-the-art provers are very fast (Figure 4.2). If backtracking is required (Figure 4.3) runtimes for ABox consistency checking increase. Note that this is not due to garbage collection, as can be seen by comparing the curve named "Cons. GC" (accumulated runtimes for garbage collection during consistency checking). ABox consistency checking can be done offline and corresponds to computing index structures in a database system.

Comparing the runtimes for query answering in Figures 4.2 and 4.3 (see the corresponding curves "Queries") reveals that backtracking does not influence query answering to a large extent (at least not in the LUBM case we investigated). The total runtime is indicated with "Total", garbage collection time is indicated with the curve "Total GC". Indeed, garbage collection adds its share to the runtime.

The results we achieve were possible with dedicated storage management techniques (e.g., offered by the implementation language Franz Allegro Common Lisp). With this basis it is possible to declare that all data structures for storing the LUBM TBox, ABox, and index structures are not examined by the garbage collector. If this is not done, garbage collection time dominate all other runtimes to a large extent. Due to the large amount of data in LUBM runtimes being examined over and over again by the garbage collector, and querying times increase in a superlinear way. The declaration of data structures as persistent (in the sense of being non-garbage) is provided after the ABox consistency check.

We take LUBM as a representative for largely deterministic data descriptions that can be found in practical applications. The investigations reveal that description logic systems can be optimized to also be able to deal with large bulks of logical descriptions quite effectively. LUBM is in a sense too simple but the benchmark allows us to study the data description scalability problem.

We argue that the concept rewriting technique is advantageous not only for RacerPro but also for other tableau-based systems. Future work will investigate optimizations on large ABoxes and more expressive TBoxes. Our work is based on the thesis that for investigating optimization techniques for more expressive TBoxes, we first have to ensure scalability for TBoxes such as

those that we discussed in this chapter. We have shown that also for instance retrieval, scalability can be achieved with tableau-based reasoners. Note that optimizations, for instance, for qualified number restrictions are not known for other approaches to query answering (e.g., resolution-based approaches).

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] S. Bechhofer, I. Horrocks, and D. Turi. The OWL instance store: System description. In *Proceedings CADE-20*, LNCS. Springer Verlag, 2005.

[3] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.

[4] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Boook*. Prentice Hall, 2092.

[5] 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.

[6] 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)*, LNCS. Springer Verlag, 2004.

[7] V. Haarslev and R. Möller. Optimization techniques for retrieving resources described in OWL/RDF documents: First results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004, Whistler, BC, Canada, June 2-5*, pages 163–173, 2004.

[8] V. Haarslev, R. Möller, and A.Y. Turhan. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. 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.

[9] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[10] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. Technical report, University of Manchester, 2006.

[11] M. Wessel and R. Möller. A high performance semantic web query answering engine. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.

[12] Z. Zhang. Ontology query languages for the semantic web: A performance evaluation. Master's thesis, University of Georgia, 2005.

Part III

Query Languages and Spatial Reasoning

Chapter 5

Ontology-based Information Systems

It is now widely accepted that *ontologies* will play an important role for the next generation of information systems (ISs). The use of ontologies for ISs will not only enable "better" and "smarter" retrieval facilities than current ISs based on the predominant relational data model (see the vision of the Semantic Web, [1]), but also play a key role for achieving data and information quality (e.g., the data model or data itself can be checked for consistency, redundancy, ...), information interoperability and integration (e.g., formal definition of vocabulary and information source mappings, ...) [2, 3, 4]. Ontologies can play a key role in solving problems raised by *semantic heterogeneity* in ISs based on different conceptual and/or logical data models, since ontologies inherently work on a semantic rather than on a syntactic level and thus allow to encode and incorporate many of the conceptual domain constraints into the IS machinery [5].

In this chapter we present a formal and implemented generic framework for building *Ontology-Based Information Systems (OBISs).* As such, our framework must offer means for 1. the extensional layer, 2. the intensional layer, and 3. the query component. Our framework is heavily influenced by *Description Logics (DLs)*, but in addition offers pragmatic solutions for certain problems we have encountered during the endeavor of realizing OBISs with standard *Description Logic Systems (DL systems).* We make these problems transparent by means of a case study: Design and implementation of an ontology-based GIS on the basis of a description logic system. The ABox is the *extensional component*, representing the actual "database" or information store in terms of so-called *assertions*. From a first-order logic perspective, the ABox contains closed – and in most cases even atomic – ground formulas (also called facts). The TBox is also called the *intensional component* and contains the *terminology,* ontology, schema, ... From a first-order logic perspective, it contains closed universal first order sentences (axioms).

We discuss and present the pragmatic solutions for that IS domain in our framework. One main focus in this case study is on *ontology-based query answering.* The DLMAPS *system* realizes ontology-based spatio-thematic query answering (see below) for city maps [6, 7].

Most retrieval systems nowadays still use rather simple thesaurus-based retrieval models (or statistical models, which are not considered here at all). However, since the advent of automated theorem proving, several prototypical "intelligent" retrieval systems have been built [8, 9, 10].

Let us first provide some background. An *ontology* provides the vocabulary of a conceptualization in a machine-processable and "-understandable" (e.g., logic-based) format such that the inherent domain constraints and their interrelationships are not lost.[1] According to Gruber [11], an ontology *is an explicit formalization of such a conceptualization;* formal means that the conceptualization is given in a machine-processable language with formal semantics so that the semantics of the conceptualization is available to the machine. The term *semantic information processing* describes this situation quite accurately, although one can claim that information is "semantic" *per se. Ontology-based query answering* then means that the (defined) vocabulary from the ontology

[1] Unlike in the relational model, where there is usually an information loss going from the conceptual to the logical data model, e.g., cardinality constraints in the ER diagram are no longer found in the table declarations, etc.

can be used in queries to retrieve the desired information (by means of query answers) from the extensional IS component. The *query answering engine* is responsible for computing these answers.

DLs are nowadays an accepted standard for decidable knowledge representation. It can also be claimed that DLs provide the *theoretical foundation* for (formal) ontologies, as well as for the Semantic Web (e.g., the *Ontology Web Language,* OWL is inspired by DLs, and OWL DL is somehow just another syntax for the DL called $\mathcal{SHOIN}(\mathcal{D}n)$).

It is likely that DL systems (FaCT^{++}, RacerPro, ...), and DL-based technology in general, will play an increasingly important role for building the next generation of these deductive ontology-based information systems (as well as for the Semantic Web). Still, the number of implemented OBIS is rather small, so there is not much experience available. This is not surprising, since current DL systems still have a long way to go. Let us discuss *some problems with todays DL technology.*

The RacerPro DL system [12] implements the very expressive DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, also known as $\mathcal{SHIQ}(D^-)$ [13, 14], offers ABoxes as well as concrete domains ("datatypes"). RacerPro can be called an empirically successful system, since it is widely used and commercially distributed by the start-up company Racer Systems. In the rest of the paper we will primarily use and consider RacerPro as our standard DL system.

5.1 Problem Identification

We have identified 7 main problems **P1**–**P7** which contribute to the difficulties encountered regarding DL system usage for OBIS:

On the one hand, DLs themselves have their deficiencies and are thus not a panacea for arbitrary information modeling and representation. DLs are very well suited for the representation of (and reasoning about) semi-structured (or even unknown/indeterminate) information [15], but things become more complicated if, say, n-ary relationships or special "non abstract" domains such as space are considered (→ **P1:** "DL applicability problem"). Then, either non-standard DLs or complicated logical encodings are needed. For these non-standard DLs, no working systems exist, and complicated logical encodings are likely to decrease the performance and complicate the information handling and maintenance.

Moreover, due to the the well-known "expressivity vs. complexity" tradeoff, reasoning with expressive DLs can be intractable. Thus, *data scalability* is not easy to achieve for expressive DLs. On the other hand, there exist inexpressive ontology languages like RDF(S) [16, 17, 18] which scale well w.r.t. the data (obviously, scalability must be achieved for the Semantic Web), but these approaches fail to scale w.r.t. expressivity. We believe that a generic framework for OBIS building should be parameterizable in *both* dimensions (→ **P2:** "data and expressivity scalability problem") [19, 20, 21].

On the other hand, DL systems somehow live in their own realm and are thus not really interoperable with the rest of the more conventional IS infrastructure, e.g., existing relational database technology (→ **P3:** "interoperability and middleware problem"). However, due to the inherent intellectual complexity and since building a DL system is a non-trivial task, existing DL systems must be reused and exploited as *componentware* if possible.

Even though "semantic middleware standards" such as DIG exist [22], it can be observed that for building practical OBIS significant API functionality is still missing, only part of which is currently about to be standardized in DIG$_{2.0}$ [23], e.g., cursor-based incremental query answering as in nRQL [24, 25], access to told information or the concrete domain.[2] Compared with the APIs found in *relational database management systems (RDMSs)*, still whole areas of API functionality are uncovered, e.g., functionality regarding the "physical schema" or storage layer of a DLs (definition of index structures, persistency, → **P4**: "missing storage layer functionality problem"). Currently it seems unlikely that any consensus among the designers of DLs can be reached regarding these issues. It is interesting to observe that DLs are currently mainly perceived as systems on the *knowledge level* [26], but not so much as systems on the *symbol resp. implementation level*, unlike RDMSs which provide a clear more-or-less standardized account at the symbol level. Tuning RDMSs on that level is daily bread and butter for many IT professionals. We believe that

[2] Still, the client side DIG implementations have not reached a level of matureness as more conventional middleware, e.g., Corba, which also addresses issues such as caching, load balancing, security, etc.

in RDMSs the distinction between symbol and knowledge level is much more blurred, since the relational model is somehow as much a *logical* as *a physical* data model, unlike DLs.

Moreover, as for RDMSs, plug-in mechanisms or "stored procedures" would be beneficial in order to open-up the server architectures for applications as well as to achieve high bandwidth communication. *Extensibility and Openness*, especially for the storage resp. physical layer, is still not achieved for standard DL systems (→ **P5:** "extensibility problem", see also [6, 23]).

Only recently issues such as *persistency and powerful query languages (QLs)* have been considered and incorporated into DL systems (→ **P6:** "QL and persistency problem") [27, 28, 29, 30]. However, these are indispensable for OBISs.

Even if the logic utilized for the ontology / knowledge base would in principle permit data scalability and "good performance", nowadays it is still sometimes the case that standard DL systems implementing very expressive DLs don't perform as good as dedicated provers for small and inexpressive DLs. This is not surprising, since in specialized provers more effort can be put in implementing highly specific optimizations. This effort would not necessarily be good invested for provers which implement more expressive DLs, since these have to cover a much broader language spectrum sufficiently performant. With more and more dedicated optimizations whose applicability must be automatically detected (which can be very complicated as well), software maintenance of the DL system becomes a serious problem.

This would imply that, for a given inference problem, not only a DL reasoner with sufficient expressivity should be selected, but moreover also a (optimized) DL reasoner whose *upper complexity bound* tightly matches the required expressivity (see also [31]).

From a knowledge level perspective, most expressive DL systems only have to implement one core inference algorithm (e.g., an ABox satisfiability checker), since the other inference problems are reducible to that core algorithm. However, viewed from the symbol / implementation level, this perspective is inadequate, since nowadays highly dedicated optimized inference algorithms exists, all of which have to be implemented in order to get a working system in practice. These procedures are sometimes even more complex than tableau calculi and thus *deserve a clean separation* from other parts of the system code in order to achieve maintainability (→ **P7:** "Identification of maintainable software abstractions in order to cope with the extensibility and adaptability problem"). A generic framework for OBIS building should account for these aspects.

Our solution for these problems is to provide many small and specific provers for specific problems instead of one big prover. However, a big number of small provers can only be more maintainable if appropriate software abstractions are provided. Our framework thus offers *domain specific languages (DSLs)* for prover definition that also account for reuse and adaptability of prover components, e.g. tableaux rules. These DSLs make the prover definitions nearly as concise and comprehensible as the mathematical tableaux calculi.

To sum up, the identified problems make it nowadays still hard to use DL systems simply as a "smarter" RDMS replacement for industrial strength ISs.[3]

Whereas for RDMSs, the *IS design problem* nowadays no more lies *within* the individual components resp. layers, but solely in the combination and interoperability of the layers, it is justified to say that for DL systems the problems still primarily reside *within* the layers. Thus, a broad "3 Tier Architecture" is not of much help in this setting if one considers architectures for OBIS.

5.2 Layered vs. Integrated Approaches for OBIS

Why not simply use an RDMS for the storage layer of such an OBIS? *Ontology-based query answering* requires inference. From the point of view of the RDMS, the inference algorithms then have to reside in the *application layer* (of the "3 Tier Architecture").[4] Which parts to access from the database can only be determined at runtime of the inference algorithm. Thus, queries over queries would have to be posed to the RDMS; moreover, the index structures of the RDMS will probably

[3] Even if – in most cases – this would probably result in a *miss-use of the DL system* we nevertheless still think that a DL system should perform as well as an RDMS for "dumb bulk data", otherwise the technology will not be commonly accepted in the long run.

[4] It is unrealistic to assume that a system as complex as a tableaux reasoner can be realized as a stored procedure within a RDMS.

not be of great help. Due to the layer separation, there is a lot of communication overhead. Obviously, it would be better if the computation and integration of required sub-results could be done in the RDMS itself by means of a single query. However, again this is only possible for *inexpressive DLs* . For example, in the QuOnto system, ontology-based query answering can be performed by the RDMS query answering engine solely, since the inexpressivity of the exploited DL makes it possible to expand the original query in such a way that it takes the ontology vocabulary / TBox into account [32]. Thus, no real ABox retrieval algorithms are needed. So, this so-called *layered approach* for OBIS has some drawbacks since it does not account for expressivity scalability.

We therefore pursue mainly an *truly integrated approach* for OBIS, although the implementation burden is very high then. Truly integrated means that inference algorithms and query evaluation engines are within one single component and share many data and index structures. However, our framework also accounts for the layered approach.

As a running OBIS example domain we use the domain of digital city maps. In this IS domain of digital city maps, we must

1. pragmatically solve the map representation problem, especially regarding the spatial and thematic aspects of the map objects, and
2. provide an expressive *spatio-thematic query language* (QL) which supports ontology-based query answering (w.r.t a "city map background ontology"). This QL must be able to address spatial as well as thematic aspects of the map objects.

This chapter provides the following contributions:

- The main contribution of this chapter is the description of the framework and of its conceptual and software-technical abstractions. Thus, this framework contains abstractions and working implementations to realize 1. the extensional component, 2. the intensional component, 3. the query language component of an OBIS. We describe how the identified problems **P1**–**P7** are tackled.

  It will become clear that for all three areas, highly flexible solutions are provided: for 1., the so-called *substrate data model;* for 2., the *substrate query language (*SuQL*) framework*; and for 3., the MiDeLoRa toolkit. The flexibility of these abstractions is demonstrated empirically by means of the instantiations (DLMAPS, nRQL, SnRQL, see below).

- Due to the intellectual inherent complexity of the field, we believe that DL system use cases / application studies are valuable per se. In the DLMAPS domain, the situation is even more complicated, due to the applicability problem for DLs, which mainly concerns the representation of (and reasoning about) the spatial aspects of the maps (which we will call the "spatial representation problem" in the following).

  Spatial representations are, *in principle,* possible with expressive spatial *concrete domains (CDs)* [33, 34] or specialized DLs [35] or spatial modal logics [36]. However, many of these logics are either undecidable, and if they are, then no mature DL system supporting these non-standard DLs exist. In principle, our framework allows for the definition of tableau provers for such specialized languages.

  However, in this paper we focus on more pragmatic representations (see below) which incorporate RacerPro. We will demonstrate various representation options for the city maps and thus document what can be done with nowadays available *standard* DLs (or Semantic Web) technology such as RacerPro in such a difficult terrain.

- Moreover, we present some important optimization techniques which are critical for ontology query answering engines.

- Finally, as part of the overall framework, we present some details of the MiDeLoRa [5] toolkit for DL system crafting. It provides high-level domain specific languages and employs fresh architectural ideas.

  As for the other software abstractions and techniques, we claim that these can be useful for other developers of related systems.

[5] *Mi*chael's *De*scription *Lo*gic *Rea*soner

Somehow our approach is related to JENA [37], but we have a somewhat broader scope, and the underlying knowledge resp. data models are more general than RDF(S): In contrast to other frameworks, we support (qualitative and quantitative) spatial representations, ABoxes, RDF(S)-like structures, etc. The framework contains a high performance substrate query answering engine which still has some unique features which are not found in other ontology query answering engines (see below). Moreover, especially with MiDeLoRa, our research is also in a line of research in software engineering on so-called *(software) product families*, which is currently a "hot topic".

We first describe the overall framework and explain how the identified problems **P1**–**P7** are addressed. Then comes the DLMAPS case studies: We first describe the IS domain of digital city maps, the concrete map data we use, as well as the idea of spatio-thematic ontology-based query answering on such city maps. We discuss the "spatial representation problem" and present 4 representation options for the extensional component in our framework. Next we describe the substrate QL framework SuQL, which plays a crucial role here. The nRQL ABox query language is discussed as a concrete instantiation resp. SuQL specialization. Especially for the DLMAPS system, we show how nRQL can be extended by spatial atoms so to become a spatio-thematic QL. We then describe the core features of the SuQL query answering engine and also present some indispensable optimizations and dicuss their effectiveness. Next we present the novel ideas materialized in the MiDeLoRa toolkit.

5.3 An Architectural Framework for OBIS Building

Our framework for OBIS building is designed and implemented to address the problems P1 - P7 as follows:

P1, "DL applicability problem" In the DLMAPS domain, there is a need to represent *the spatial aspects* of the maps, and for other IS domains, there may be other informational aspects which cannot be represented in a single representational framework (e.g., a DL ABox). Regarding the spatial aspects of map objects, their representation is difficult or impossible with a *standard* DL ABox (see below). Thus, our framework is based on a *generalized graph-based data model, called the substrate data model.* Substrates offer more flexibility, than, say RDF(S) graphs or ABoxes, since they can also represent, for example, digital vector maps (see below). Substrates thus provide generic extensional representation means. A DL KB (ABox+TBox) can be seen as a substrate with an *associated background theory* (see below). Moreover, substrates can be *hybrid* an thus allow for information resp. representation layering. A DL ABox can be layered with an arbitrary substrate; thus, the DL applicability problem can be defused pragmatically.

A substrate is an instance of a Clos (Common Lisp Object System, [38]) class, which can exploit inheritance. A rich set of substrate classes is already provided. On the one hand, a substrate is a representation on the knowledge level, but also – and much more importantly in this work – a structure on the symbol / implementation level. It can provide dedicated index structures, etc.

P2, "Data and Expressivity Scalability Problem" DLs thenselves form a *family* of representation languages. Thus, DLs themselves offer, in principle, *expressivity scalability* (given they are applicable for the considered IS domain). *Data scalability* can nowadays be achieved for simpler DLs (or RDF(S), ...). In order to achieve data scalability, not only the knowledge, but also the symbol level must be considered, e.g., a *database substrate* can be used if the extensional data is extensive. Thus, the framework accounts for data scalability.

It also accounts for expressivity scalability, since MiDeLoRa allows for the definition of specialized provers. *Provers* are conceived as regions in the three-dimensional so-called MiDeLoRa **space**. The MiDeLoRa space has the structure $\mathcal{S} \times \mathcal{L} \times \mathcal{T}$: The $\mathcal{S}$ axis is the *substrate class* for which the prover is defined; on the $\mathcal{L}$ axis, the *(DL) language* supported by the prover is specified, and the $\mathcal{T}$ coordinate specifies the *task class* of the prover (among others, $\mathcal{T}$ contains the set of DL *standard inference problems* [39]). Since substrates, languages and tasks are modeled as Clos classes and inheritance is exploited, a MiDeLoRa prover which is defined for a certain (S, L, T) point basically covers a region in the MiDeLoRa space. This region-covering of the space can be organized very flexible, since Clos offers so-called multi-methods which exploit late binding / polymorphism according to all three arguments (S, L, T). Although this is a simple idea, it still

makes the approach unique.

P3, "Interoperability and Middleware Problem" The substrate data model can provide an abstraction layer on top of which the OBIS can be built. This abstraction layer can for example shield the client code of the OBIS from details in the API of, different external standard DL systems, if such DL systems are used as componentware (see also the *Design Patterns Adapter, Bridge and Facade* in [40]). Moreover, the substrate can offer caching mechanisms, abstract from remote vs. local API procedure calls, etc. In short, a substrate *can* also play the role of a *mediator and/or semantic middleware.*

In principle, since a substrate is a very general graph based data model, the (remote) componentware system must not even be a DL system, but could also be an RDF(S) triple store, an RDMS on which a graph-like view is established, etc.

Since substrates are CLOS classes utilizing inheritance that implement interfaces, additional services can easily be offered by means of substrate sub-classing. For example, a "RACERPRO substrate" class will offer unique RACERPRO services in addition to the more general methods / services inherited from its more generic "DL system substrate" superclass.

P4, "Missing Storage Layer Functionality Problem" Given that a substrate is not only a conceptual data model (an abstract data type on the knowledge level) but also implemented as a CLOS class, it is obvious that, by means of programming, the framework offers the flexibility to address and parameterize the storage layer. For example, in the *SBox substrate* class we have implemented spatial index structures. It is clear that API functions for controlling such highly specific aspects in the physical layer will never become part of any future DIG proposal, since such a list of "add ons" would be open ended.

P5, "Extensibility Problem" Extensibility and openness of the architecture is obviously realized (although white box reuse – reuse in frameworks – has been identified as problematic in some cases; however, it is also known that domain specific languages can resolve some of these problems [41]).

P6, "QL and Persistency Problem" Obviously, a QL is needed in order to retrieve the information back from the substrate. The framework offers a so-called *substrate QL* SUQL which is as open, extensible and parameterizable as the rest of the framework, but still decidable. Basically, the QL framework offers unions of (grounded) conjunctive queries. Extensibility is granted since specialized *query atom classes* can be defined for substrate classes. For example, for a map substrate, query atoms can be defined which check whether certain qualitative spatial relationships hold in a geometric representation of the map. We will demonstrate that the SUQL framework is sufficiently expressive and flexible for the implementation of a pragmatic spatio-thematic QL as well as for a pragmatic ABox query language (NRQL).

A substrate can also be made persistent in a file or a in a relational database (such as MYSQL). The required COMMON LISP/CLOS serializers have been implemented by the authors.

P7, "Identification of maintainable software abstractions problem" We already described the three core notions of our OBIS framework informally: 1. the substrate data model as a flexible extensional component, 2. the substrate QL framework, and 3. the MIDELORA toolkit.

But how do we define and extend these abstractions? Obviously, their implementation (as a CLOS class) can be very hard. Thus, our framework already supplies a rich set of ready-to-use of-the-shelf classes (substrates, query language atoms, MIDELORA provers for standard DLs). We claim that in many cases our framework will already provide the required software building blocks ("components"); in many cases, only slight adjustments will be needed. Due to its *openness and adjustability* this is often possible.

In case something really new must be defined, e.g. a specialized prover or substrate class, the implementor must not start from scratch, since the framework is designed for maximal reuse. For example, if a new tableau prover for a certain DL shall be defined, then, even if the required prover is not already available and no existing prover can be adjusted, then possibly the required *standard tableaux rules* are available. As provers, tableaux rules are provided as abstractions in the framework, and there is a rich library defined. As already mentioned, DSLs are used for their definition.

The overall architecture of the framework is depicted in Fig. 5.1. The depicted subsystems will be discussed in the following. Please note that parts of this framework (namely the parts required for NRQL) have been moved into into the RACERPRO DL system. This effectively

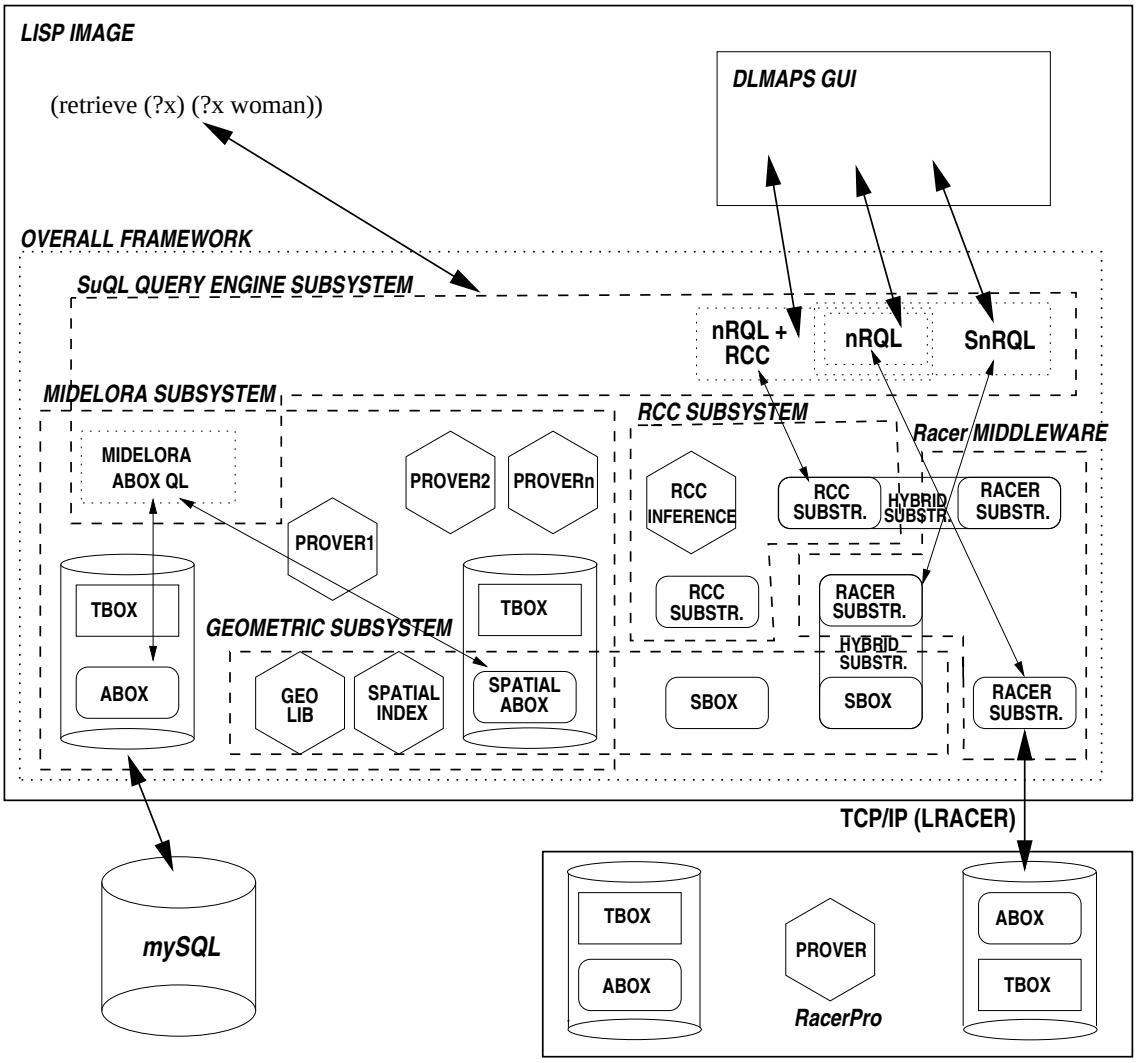

Figure 5.1: Overall Architecture of the Framework

replaces the TCP/IP-based remote procedure (function) calls shown in Fig. 5.1 with local procedure (function) calls. From a software perspective, the resulting architecture is still layered. In contrast, *tight integration* based on structure and software abstraction sharing is indicated by means of non-empty subsystem boundary / region intersections in that diagram.

5.3.1 The Substrate Data Model and Substrate Query Language at a Glance

Formally, we base our framework on the semi-structured graph-based *substrate data model* which provides the required flexibility and extensibility for the extensional component. As explained, the substrate model serves both as a mediator and abstraction layer ("semantic middle ware"), enables us to specify and build extensional representation layers for spatial and/or hybrid representations (e.g., for the DLMAPS system), etc. Substrates are general enough to encompass ABoxes and RDF(S) graphs. As such, it is not surprising that a substrate is defined as a very general notion:

Definition 1 A *substrate* is an edge- and node-labeled directed graph $(V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$, with V being the set of *substrate nodes*, and E being a set of *substrate edges*. The node labeling function $L_V : V \rightarrow \mathcal{L}_\mathcal{V}$ maps nodes to descriptions in an appropriate node description language $\mathcal{L}_\mathcal{V}$, and likewise for $L_E : E \rightarrow \mathcal{L}_\mathcal{E}$, where $\mathcal{L}_\mathcal{E}$ is an edge description

language. ■

The languages $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are not fixed and can be seen as *(variable-free denoted) subsets of first-order predicate logic, FOPL* (e.g., modal logic, description logic, propositional logic, ...). Using this FOPL perspective, V is a set of constant symbols, and L_V and L_E are *indexing functions* into sets of closed FOPL formulas.

Let us illustrate this with an example. Consider an $\mathcal{ALC}$ ABox $\mathcal{A}$. We can consider this ABox as a substrate $S = (V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$ if we identify V with the ABox individuals, $V = \mathsf{inds}(\mathcal{A})$, E with the set of pairs of individuals mentioned as arguments in role assertions, $E = \{\,(i,j) \mid\ (i,j) : R \in \mathcal{A}\,\}$, with $\mathcal{L}_\mathcal{V} = \mathcal{ALC}$, and $\mathcal{L}_\mathcal{E} = (\mathcal{N}_\mathcal{R}, \sqcap)$ would be the set of $\mathcal{ALC}$ role names $\mathcal{N}_\mathcal{R}$ closed under conjunction, such that $C \in L_V(i)$ iff $i : C \in \mathcal{A}$, and $R_1 \sqcap \cdots \sqcap R_n = L_E((i,j))$ iff $\{\,R_1, \ldots, R_n \mid\ R_i \in \mathcal{N}_\mathcal{R}, (i,j) : R_i \in \mathcal{A}\,\}$. From the FOPL perspective, this would simply result in a set of formulas of the form $\{\Phi(C)_{x \leftarrow i}, \ldots, R_1(i,j), \ldots, R_n(i,j)\}$, where $\Phi(C)$ returns the FOPL translation of the concept C (a first order formula with one free variable, x, e.g. $\Phi(\exists R.C) = \exists y R(x,y) \wedge C(y)$, [39]). However, for many substrates, the corresponding FOPL set will simply contain *ground atoms (facts).*

An *associated TBox of an ABox* manifests itself in additional FOPL sentences. Formally, we would simply define a substrate with a background theory (having an additional set of closed FOPL axioms). This should be clear. We will give an example for such a background theory when we discuss the RCC substrate. However, additional sets of axioms can also assumed to be *intrinsically encoded* into the substrate structure as well, without explicit representation as sentences, see below.

To get spatial representations resp. substrate, we simply state that a substrate can also encode geometric / spatial structures by FOPL means, as needed. For the DLMAPS system, we assume that the nodes are instances of spatial datatype (polygons, points, lines, ...). Such a geometric substrate is called an *SBox (Space Box).* We simply state here that the geometry of such spatial nodes can be described using an appropriate *(FOPL-based) geometry description language.* It does not add to the message of this chapter to go into formal definitions here.

Unlike for an ABox, it is reasonable to assume for an SBox that its *logical theory is complete,* i.e., that the *Closed World Assumption (CWA)* is used. Then, there is neither underspecified nor indeterminate information in an SBox; it simply represents "spatial data". Viewed as a set of FOPL ground atoms, the SBox is basically isomorphic to its (unique minimal) Herbrand model then. On the declarative knowledge / sentence level we can simply assume that the well-known *Clark completion axioms* are present [42], and that their impact will be "intrinsicly" encoded into the inference procedures defined for an SBox, on the symbol level.

Note that, since we simply rely on standard FOPL semantics, everything is well defined, since we simply inherit the standard FOPL notions of satisfiability, entailment ("$\models$"), etc. We need the entailment relationship for the SuQL.

Some words in defense of the model: We do *not* claim that this data model is interesting from a theoretical perspective. Its generic character is of course also its weakness. Thus, it must be specifically *instantiated* (e.g., ABox or SBox). However, the given definition enables a formal specification of the semantics of the framework, and within the framework it serves as a very useful abstraction, as will become clear.

The data model is also *somehow* inspired by the work on $\mathcal{E}$-Connections [43] or the tableaux data structure [13], as well as by RDF(S). However, it would be inappropriate to claim that this is an $\mathcal{E}$-Connection application, since we are basically just using labels defined by means of first order logic, and similar knowledge models are used in AI since the 1960s.

Note also that, from the implementation / symbol-level perspective, the substrate perspective of an ABox can be "virtual", i.e., the API functions of the substrate mediation layer can just pass through to the API functions of the ABox reasoner, e.g. RacerPro, that is maintaining the ABox physically (or an RDF(S) triple store, ...). In principle, such a "virtual RacerPro substrate" would be sufficient for the nRQL implementation. However, in order to maximize parallelism of query answering the RacerPro substrate is not completely virtual, since it implements dedicated caches and index structures needed for performant query answering on RacerPro ABoxes (see below).

A substrate is implemented as a Clos class. Since Clos offers multiple inheritance, it becomes possible to define *combinations of substrates.* For example, MiDeLoRa offers so-called *spatial*

ABoxes. This class mentions both the class ABox as well as the class SBox as superclasses. As a result, nodes in such spatial ABoxes are, on the one hand, *ABox individuals*, and instances of spatial data types on the other hand [6]. In case of MiDeLoRa, this eliminates the need for a hybrid representation in favor of an integrated representation.

In the BOEMIE project, an information system about sports events such as, e.g., marathon events in large cities, are considered as an example application. For such an information system which involves queries w.r.t. spatial, temporal and terminological information it is appropriate to provide specialized query language as well as specialized reasoners.

The SuQL framework enables the creation of specialized substrate QLs, tailored for special substrate classes (ABoxes, SBoxes, ...). The SuQL framework is based on the general notion of *(ground) query atom entailment*. All what matters here is that a *notion of logical entailment* ($\models$) between a substrate S and a *query atom for* S is defined and decidable. Query atoms are, conceptually slightly simplified, again FOPL formulas with one resp. two free FOPL variables (we use x and y in the remaining paper for these). We talk of unary resp. binary query atoms. Thus, given the unary atom P and binary atom Q, $i, j \in V$, then $S \models P_{x \leftarrow i}$ and $S \models Q_{x \leftarrow i, y \leftarrow j}$ must be well-defined and decidable. That's all.

The SuQL framework provides a great deal of flexibility, extensibility and adaptability, since specialized query atoms (resp. P and Q) can be tailored for specific substrate classes, e.g., if S is an SBox, then P, Q can be spatial predicates (e.g., RCC predicates, see below). If only a notion of entailment is defined and decidable for these specialized atoms, then the substrate query answering engine immediately supports the evaluation of these atoms. Of course, in many cases dedicated optimization and index structures are needed for efficient query evaluation. Specific index structures must be provided by the substrate. Again, the *cost-based* SuQL *query optimizer* is easily configurable by means of method overwriting.

Since the substrate as well as the query atoms are instances of Clos classes, the $\models$ relation is implemented as a (binary) Clos multi-method `substrate-entails-atom-p`. This is a simple idea, but it demonstrates the ingenious power of Clos which simplifies and conceptually clarifies the design of the framework tremendously. In the definition of such a `substrate-entails-atom-p` method, also inherently encoded axioms can be taken into account, simply by means of programming. For example, the Clark completion axioms must not be explicitly present as sentences. They are only needed for a description of the semantics on the knowledge level, but not on the symbol level. However, in many cases, a declarative inference algorithm will be called, e.g. a MiDeLoRa prover or a RacerPro API function.

However, the previous statements are not meant to reopen the "declarative vs. procedural" debate. Instead, our framework shows that both approaches can live together well, provided an embedding into a uniform, formally profound framework such as ours can be given.

5.4 DLMAPS: Ontology-Based Queries to City Maps

We now describe the digital city maps scenario. As mentioned, we are primarily using RacerPro as our standard DL component reasoner, but other setups are possible as well (some of these are described in the following).

5.4.1 The DISK Data

We are using digital vector maps from the city of Hamburg provided by the land surveying office ("Amt für Geoinformation und Vermessungswesen Hamburg"); these maps are called the DISK ("Digitale Stadtkarte"). Part of the DISK is visualized by the Map Viewer component of our system in Fig. 5.2. Each map object (also called *geographic feature*) is *thematically annotated*. The basic *thematic annotations (TAs)* have been established by the land surveying office itself. These TAs say something about the "theme" or semantics of the map objects. Simple concept names such as "green area", "meadow", "public park", "lake" are used. A few hundred TAs are used and documented in a so-called *thematic dictionary (TD)*, which is GIS-typically organized in so-called (thematic) *layers* (e.g., one layer for infrastructure, one for vegetation, etc.).

Sometimes, only highly specific TAs are available, such as "Cemetery for non Christians", and generalizing *common sense vocabulary*, e.g. "Cemetery", is missing. This is unfortunate, since it

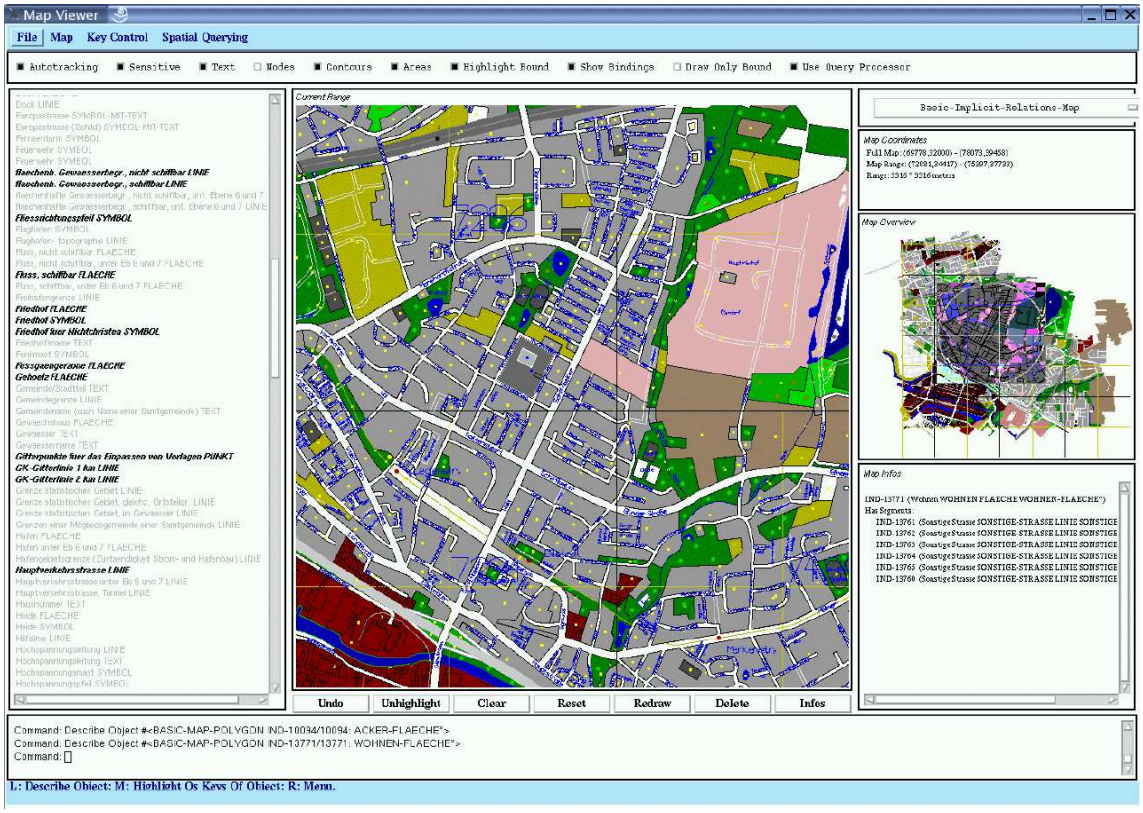

Figure 5.2: The MAP VIEWER of the DLMAPS System

prevents the intuitive usage of common sense natural language vocabulary for query formulation, especially for non casual users. We have repaired this defect by adding a background ontology (in the form of a TBox) providing generalizing TAs by means of taxonomic relationships.

On the other hand, *defined concepts* ("if and only if") can be added and exploited to *automatically enrich* the given basic annotations. Thus, we might define our own needed TA "public park containing a lake" as a "park which is public which contains a lake" with a TBox axioms such as

$$public\_park\_containing\_a\_lake \doteq park \sqcap public \sqcap \exists contains.lake$$

or

$$bird\_sanctuary\_park \doteq park \sqcap \forall contains.\neg building$$

and we might want to retrieve all instances of these concepts. This means that such instances must be recognized automatically, and this is what ontology-based query answering is all about. Obviously, inference is required to obtain these instances, since there are no *known* instances of *public_park_containing_a_lake*. For simple queries, simple *instance retrieval queries* might be sufficient. However, for reasons of expressivity and because we want to retrieve *constellations*[6] *of map objects* a QL with variables is needed whose answer bindings resp. "answer tuples" can be displayed meaningfully and visualized as in Fig. 5.3.

A definition such as *public_park_containing_a_lake* refers to *thematic as well as to spatial aspects* of the map objects:

Thematic aspects: the name of the park, that the park is public, the amount of water contained in the lake, etc.

Spatial aspects: the *spatial attributes* such as the area of the park (or lake), the concrete shape, qualitative *spatial relationship* such as "contain", quantitative (metric) spatial relationships such as the distance between two objects, etc.

[6]We use the term "constellation" to stress that a certain spatial arrangement of map objects is requested with a query.

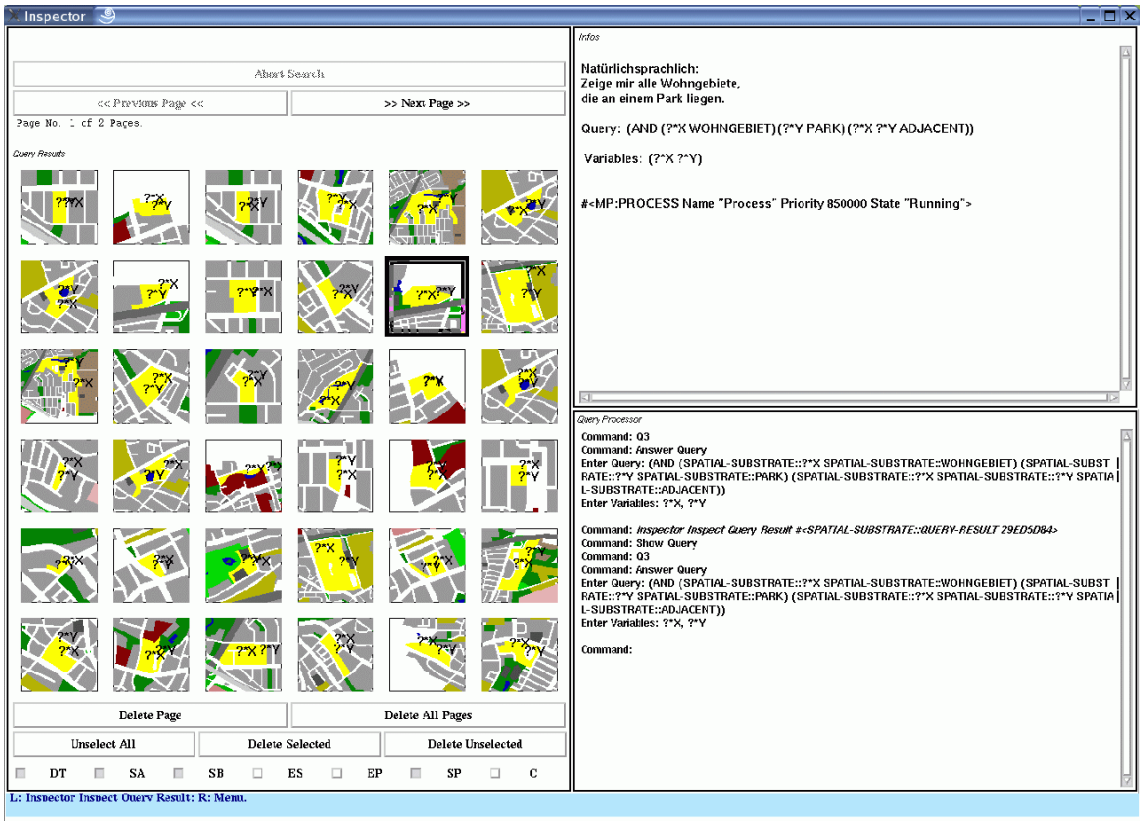

Figure 5.3: The Query Inspector of the DLMAPS System

We use the following terminology: a *thematic concept* refers only to thematic aspects, whereas a *spatial concept* refers solely to spatial aspects. A *spatio-thematic concept* refers to both. In the same sense we are using the terminology *thematic*, *spatial* and *spatio-thematic queries*. A strict separation might be difficult sometimes.

Thus, there are different thematic and spatial aspects one would like to represent in the extensional component and subsequently query with a spatial QL. Since the concrete geometry is given in the map, the spatial aspects of the map objects are in principle intrinsically represented and available. This mainly concerns the spatial relationships which are depicted in the map. However, also spatial attributes such as the area or length of a map can in principle be derived (computed from the geometry, although this will not be very accurate. A function that exploits the map geometry to compute or verify a certain spatial aspect (for example, if a certain qualitative relationship holds between two map objects) is called an *inspection method* in the following. This notion is defined sufficiently precise as follows:

Definition 2 (Inspection Method) Let S be an SBox, and P be a spatial FOPL formula without free variables (for example, an RCC ground atom such as $EC(a, b)$, where $a, b \in V$). An *inspection method* is a (geometric) algorithm which exploits the *geometry of* S to decide whether $S \models P$ holds. ■

It is obvious that *qualitative* spatial descriptions are of great importance. On the one hand, they are needed for the definitions of concepts in the TBox such as "public park containing a lake". On the other hand, they are needed in the spatio-thematic QL ("retrieve all public parks containing a lake"). A popular and well-known set of qualitative spatial relationships is given by the RCC8 relations [44], see Fig. 5.4.

On the other hand, since the concrete geometry is given by means of the map, in principle, *no qualitative representation is needed* in the extensional component, since it can be reconstructed at query answering time by means of inspection methods. However, if we want to use a (standard-DL)

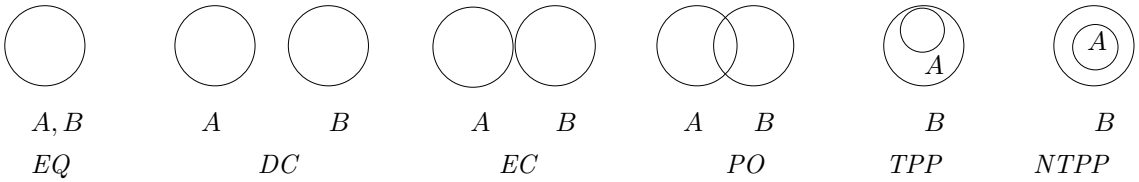

Figure 5.4: RCC8 base relations: $EQ = Equal$, $DC = Disconnected$, $EC = Externally\ Connected$, $PO = Partial\ Overlap$, $TPP = Tangential\ Proper\ Part$, $NTPP = Non\text{-}Tangential\ Proper\ Part$. All relations with the exception of TPP and $NTPP$ are symmetric; the inverse relations of TPP and $NTPP$ are called $TPPI$ and $NTPPI$.

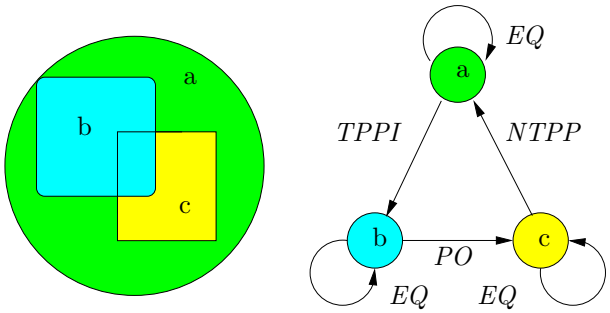

Figure 7.4(a): Concrete geometric scene and corresponding RCC8 network (inverse edges omitted)

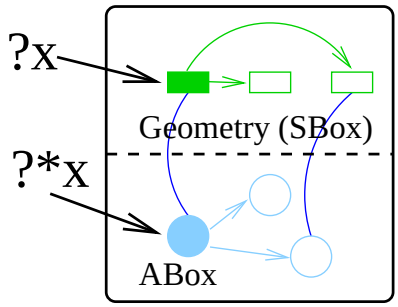

Figure 7.4(b): Hybrid map substrate: variables are bound in parallel

ABox for the extensional component, then the spatial representation options are limited, and we must primarily resort to qualitative descriptions.

5.5 Representing and Querying the DISK

It is clear that the kind of representation we will devise for the DISK in the extensional component also determines what and how we can query. Without doubt, the thematic aspects of the DISK map objects can be represented satisfactory with a standard DL. To solve the spatial representation problem of the DISK in the extensional component, we consecutively consider four different representation options and analyze their impacts.

5.5.1 Representation Option 1 – Simply Use an ABox

We can try to represent "as many spatial aspects as possible" in the ABox, given the DL supported by the exploited DL system, e.g. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ in case of RacerPro. Regarding the spatial relationships, we can only represent *qualitative* relationships. We can compute a so-called *RCC network* from the geometry of the map and represent this by means of *RCC role assertions* in the ABox, e.g. $(i, j) : TPPI$ etc. In Fig. 5.5(a) a "geometric scene" and its corresponding RCC8 network is depicted. Such a network will always take the form of a an edge-labeled complete graph, a so-called K_n (this is a standard notion in graph theory), due to the *JEPD property* of the RCC base relations. In this case, base relations are *j*ointly *e*xhaustive and *p*airwise *d*isjoint). Moreover, an RCC network derived from a geometric scene will always be *RCC consistent* (see below).

Selected spatial attributes such as area and length can be represented in the ABox utilizing the concrete domain by means of concept assertions such as $i : \exists(has\_area).\ =_{12.345}$.

Since the represented spatial aspects are accessible to RacerPro, this enables / supports richer spatio-thematic concept definitions in the TBox, for example

$$public\_park\_containing\_a\_lake \doteq park \sqcap public \sqcap \exists contains.lake$$

(we are using $\exists contains.lake$ instead of $(\exists TPPI.lake) \sqcup (\exists NTPPI.lake)$, the framework recognizes these qualitative spatial relationships and rewrites the query accordingly). Obviously, an

individual i in the ABox can only be recognized as an instance of that concept if RCC appropriate RCC role assertions are present as well.

In principle, the specific properties of qualitative spatial (RCC) relationships resp. (RCC) roles cannot be captured completely within $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (we will elaborate on this point below when we discuss qualitative spatial reasoning with the RCC substrate). This means that the computed taxonomy of the TBox will not correctly reflect the intended subsumption relationships. However, MiDeLoRa also supports $\mathcal{ALCI}_{\mathcal{RCC}}$ [35, 45]. Even though this DL is undecidable [36], the prover has been successfully applied for computation of taxonomies for $\mathcal{ALCI}_{\mathcal{RCC}8}$ TBoxes. Moreover, the deduced *implied subsumption relationships* can be made syntactically explicit by means of additional TBox implication axioms, and this augmented TBox can be used instead of the original one in RacerPro.

Much more important in our scenario is the observation that *ontology-based query answering* can still be achieved in a way that correctly reflects the semantics of the spatial (RCC) relationships with RacerPro. Consider the *instance retrieval query public_park_containing_a_lake*$(?x)$ on the ABox

$$\mathcal{A} = \{i : park \sqcap public, k : lake, j : meadow, (i,j) : TPPI, (j,k) : NTPPI \ldots\}$$

Since this ABox has been computed from the concrete geometry of the map, it must also contain $(i,k) : NTPPI$, because a RCC network which is computed from a spatial constellation that shows $(i,j) : TPPI$ and $(j,k) : NTPPI$ must necessarily also show $(i,k) : NTPPI$. In order to retrieve the instances of *public_park_containing_a_lake*, we consider and check each individual separately. Let us consider i. Verifying whether i is an instance of *public_park_containing_a_lake* is reduced to checking the unsatisfiability of $\mathcal{A} \cup \{(i,k) : NTPPI\} \cup \{i : \neg public\_park\_containing\_a\_lake\}$, or

$$\begin{aligned}\mathcal{A} \cup\ & \{(i,k) : NTPPI\} \cup \\ & \{i : (\neg park \sqcup \neg public \sqcup ((\forall NTPPI.\neg lake) \sqcap (\forall TPPI.\neg lake)))\}\end{aligned}$$

This ABox is unsatisfiable; thus, i is a *public_park_containing_a_lake*.

Regarding query concepts that contain or imply a *universal role or number restriction,* we can answer queries completely only if we turn on a "closed domain reasoning mode". We must *close the ABox w.r.t. the RCC role assertions* and *enable the Unique Name Assumption (UNA)*[7] in order to keep the semantics of the RCC roles.

To close the ABox $\mathcal{A}$ w.r.t. the RCC role assertions, we count the number of RCC role successors of each individual for each RCC role: for $i \in \mathsf{individuals}(\mathcal{A})$ and the RCC role R, we determine the number of R-successors $n = |\{\, j \mid\ (i,j) : R \in \mathcal{A}\,\}|$ and add the so-called *number restrictions* $i : (\leq_n R) \sqcap (\geq_n R)$ to $\mathcal{A}$. This concept assertion is satisfied in an interpretation $\mathcal{I}$ iff $n = \{\, x \mid\ (i^{\mathcal{I}}, x) \in R^{\mathcal{I}}\,\}$; thus, i must have exactly n R successors in every model. In combination with the *Unique Name Assumption (UNA),* this turns on a closed domain reasoning on the individuals which are mentioned in the RCC role assertions and thus prevents the reasoner from the generation of "new anonymous RCC role successors" in order to satisfy an existential restriction such as $\exists NTPPI.lake$. In order to satisfy $\exists NTPPI.lake$, the prover must thus necessarily *reuse* one of the existing RCC role fillers from the ABox [6]. Let us demonstrate this technique using the query concept

$$bird\_sanctuary\_park \doteq park \sqcap \forall contains.\neg building.$$

Assuming that both *lake* and *meadow* imply $\neg building$, we can show that i is an instance of a *bird_sanctuary*, since the ABox

$$\begin{aligned}\mathcal{A}\ \cup\ & \{(i,k) : NTPPI\} \cup \\ & \{i : (\leq_1\ TPPI) \sqcap (\geq_1\ TPPI), i : (\leq_1\ NTPPI) \sqcap (\geq_1\ NTPPI), \ldots\} \cup \\ & \{i : (\neg park \sqcup ((\exists TPPI.building) \sqcap (\exists NTPPI.building)))\}\end{aligned}$$

is again unsatisfiable, because the alternative $i : \neg park$ immediately produces an inconsistency. Thus, the alternative $i : (\exists TPPI.building) \sqcap (\exists NTPPI.building)$ is considered. Due to $i : (\leq_1\ TPPI) \sqcap (\geq_1\ TPPI)$, only j can be used to satisfy $\exists TPPI.building$, and only k to satisfy $\exists NTPPI.building$. Since $j : meadow$ and thus $j : \neg building$, $k : lake$ and thus $k : \neg building$, the ABox must be unsatisfiable.

[7] The UNA enforces that different individuals i, j are interpreted as different domain individuals in the Tarski-interpretation: $i^{\mathcal{I}} \neq j^{\mathcal{I}}$.

Thus, we have argued that spatio-thematic ontology-based query answering can be done on such an ABox representation of the DISK, and that this is to some extent – using some logical encoding tricks – possible *even with simple instance retrieval queries.*

5.5.2 Using an Expressive ABox Query Language

We now demonstrate that the RacerPro ABox query language nRQL (see below for more details) offers valuable additional query forumlation facilites in this scenario. For now, we are using grounded conjunctive queries in mathematical (Horn-logic like) syntax and assume that the reader has an intuitive understanding (in addition to our explanations). The semantics of SuQL resp. nRQL will be defined formally later.

nRQL is especially useful in this scenario, since it offers expressive means for *negation as failure.* This achieves a great deal of differentiation possibilities for query formulation: For example, we can *query for living areas adjacent to parks which contain a lake ...*

1. *... which are* provable not *adjacent to industrial areas.* Thus, all adjacent areas are provable not industrial areas (note that *adjacent* is recognized as as synonym for *EC*):

$$\begin{aligned} &ans(?living\_area, ?park, ?lake) \leftarrow \\ &\quad living\_area(?living\_area), park(?park), \\ &\quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\ &\quad (\forall adjacent.\neg industrial\_area)(?living\_area) \end{aligned}$$

2. *... for which there are no adjacent industrial areas known* (NAF negation):

$$\begin{aligned} &ans(?living\_area, ?park, ?lake) \leftarrow \\ &\quad living\_area(?living\_area), park(?park), \\ &\quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\ &\quad \backslash(\exists adjacent.industrial\_area(?living\_area)) \end{aligned}$$

   Slightly simplified, the subquery $\backslash(\exists adjacent.industrial\_area(?living\_area))$ first retrieves the instances of the concept $\exists adjacent.industrial\_area$, and then simply builds the complement set (this explains the use of "\"). Thus, a candidate binding for $?living\_area$ must be in that complement set. Please note that the instances of $\forall adjacent.\neg industrial\_area$ form a subset of this set.

3. *... for which there are no explicitly present adjacent industrial areas known:*

$$\begin{aligned} &ans(?living\_area, ?park, ?lake) \leftarrow \\ &\quad living\_area(?living\_area), park(?park), \\ &\quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\ &\quad \backslash(\pi(?living\_area) \;\; adjacent(?living\_area, ?i), industrial\_area(?i)) \end{aligned}$$

   The subquery $\backslash(\pi(?living\_area) adjacent(?living\_area, ?i)$ returns the complement set of the answer to the query
   $ans(?living\_area) \leftarrow adjacent(?living\_area, ?i), industrial\_area(?i))$
   (π is called the projection operator, see below). So, (a binding for) $?living\_area$ is in $\backslash(\pi(?living\_area) adjacent(?living\_area, ?i)$ iff for $?living\_area$ there is no explicit adjacent industrial area present. However, $?living\_area$ might have an *implicit* adjacent industrial area – thus, this query returns a *superset* of $\backslash(\exists adjacent.industrial\_area(?living\_area))$.

It is clear that (1) (classical negation) is a much too strong requirement in many queries, since it is rarely the case that two concepts can be proven to be disjoint (in most cases, explicit disjointness declarations etc. must be added to the TBox, and this does not really add to the modeling).

5.5.3 Drawbacks of the ABox Representation

Even though ontology-based query answering is somehow possible using the just discussed ABox representation, it nevertheless has the *following drawbacks:*

1. The size of the generated ABoxes is huge. Since the RCC network is explictly encoded in the ABox, the number of required role assertions is quadratic in the number of map objects, $|V|^2$ (several million role membership assertions for the DISK).

2. Most spatial aspects cannot be handled that way. For example, distance relations are very important for map queries. It is thus not possible to retrieve all subway stations within a distance of 100 meters from a certain point.

3. Query processing will not be efficient. More efficient query processing can be done if spatial index structures are added.

4. In the DLMAPS system, the geometric representation of the map is needed anyway, at least for presentation purposes. Thus, from a non-logical point of view, the ABox cannot be the only representation used in the extensional component of such a system. Thus, it seems plausible to exploit this representation for query answering as well.

5. Most importantly, we have demonstrated that this kind of ontology-based query answering works only if the domain is "RCC closed". However, DL systems are not really good at closed domain reasoning, since the *Open Domain Assumption (ODA)* is made in DLs. In contrast, since the geometry of the map is completely specified, there is neither unknown nor underspecified spatial information. This motivates the classification of such a map as spatial *data*. Thus, let us switch to a hybrid representation incorporating an SBox.

5.5.4 Representation Option 2 – Use a Map Substrate:

Due to the problems with spatio-thematic concepts and since closed domain reasoning is anyway all that we can achieve here, it seems more appropriate to represent the spatial aspects *primarily* in the SBox (a kind of "spatial database"), and *associate* an ABox with that SBox. We already mentioned that the geometry of the map must be represented in the extensional component anyway (at least for presentation purposes). If we say that the spatial aspects are *primarily* represented in the SBox, then this does *not* necessarily exclude the (additional) representation possibilities of dedicated spatial aspects in the ABox as just discussed.

The resulting hybrid $(SBox, ABox)$ representation is illustrated in Fig. 5.5(b), we call it a *map substrate*. The Figure illustrates that some ABox individuals have corresponding instances in the SBox, and vice versa. A partial and injective mapping function "$*$" which maps nodes in the SBox to nodes in the ABox (and vice versa, $*^{-1}$) is used.

Thus, we first define a *hybrid substrate* as follows:

Definition 3 A *hybrid substrate* is a triple $(S_1, S_2, *)$, with S_i, $i \in \{1, 2\}$ being substrates $(V_i, E_i, L_{V_i}, L_{E_i})$ using $\mathcal{L}_{\mathcal{V}i}$ and $\mathcal{L}_{\mathcal{E}i}$, $*$ being a partial and injective function $* : V_1 \mapsto V_2$. ■

A *map substrate* is simply a special hybrid substrate:

Definition 4 A *map substrate* is a hybrid substrate $(S_1, S_2, *)$, where S_1 is an SBox, and S_2 is an ABox (substrate). ■

If the spatial aspects of the DISK are now *primarily* kept in the SBox, then they are no longer necessarily available for ABox reasoning and retrieval. Thus, nRQL (or instance retrieval) queries are no longer sufficient to address these spatial aspects – we will thus extend nRQL to become a hybrid spatio-thematic QL, offering also *spatial query atoms* to query the SBox: SnRQL. The SnRQL query answering engine will combine the retrieved results from the SBox with results from the ABox. The thematic part of such a SnRQL query is given by a plain nRQL query, and the spatial part utilizes spatial query atoms which are evaluated on the SBox by means of inspection methods. Since the SBox represents the geometry of the map, it can evaluate the requested spatial

aspects on the SBox "on the fly" during query evaluation by means of inspection methods. The SBox provides a *spatial index*, supporting the efficient computation of spatial relationships by means of spatial selection operations. Computed spatial aspects can also be made explicit and "materialized" in order to avoid repeated re-computation (e.g., computed RCC relations can be materialized as edges).

Given a hybrid substrate, a *hybrid query* now contains two kinds of query atoms: Those for S_1, and those for S_2. In order to distinguish atoms meant for S_1 from atoms meant for S_2, we simply prefix variables in query atoms for S_2 with a "?∗" instead of "?"; the same applies to individuals. Intuitively, the *bindings* which will be established for variables must also reflect the $*$-function: If $?x$ is bound to $i \in V_1$, then $?*x$ will automatically be bound to $*(i) \in V_2$ (if defined), and vice versa (w.r.t. $*^{-1}$). Such a binding is called $*$-*consistent*. We will only consider such $*$-consistent bindings. The notion of a $*$-consistent bindings is also depicted in Fig. 5.5(b).

Assume we use a map substrate for the DISK representation now. Let us consider the example query given in Section 5.5.2 again. Since the RCC network is now no longer available in the ABox, the SBox must be queried for spatial relationships. Moreover, since the SBox uses the CWA, we can no longer answer query (1) and (2), only (3) from Section 5.5.2 has a "SnRQL equivalent" which looks as follows. Note that nRQL query atoms now use $*$-prefixed variables, since the ABox is S_2, and the SBox is S_1:

$$
\begin{array}{l}
ans(?living\_area, ?park, ?lake) \leftarrow \\
\quad living\_area(?*living\_area), park(?*park), \\
\quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\
\quad \backslash\ (\ \pi(?living\_area)\ (\ adjacent(?living\_area, ?industrial\_area), \\
\qquad\qquad\qquad\qquad industrial\_area(?*industrial\_area)))
\end{array}
$$

Thus, we do not only win, but also loose something here (queries (1) and (2) cannot be expressed). This is an important insight. On the winning side we are now able to define and evaluate much richer spatial predicates and incorporate them in SnRQL (e.g., distance query atoms, see below).

5.5.5 Representation Option 3 – Use a Spatial MiDeLoRa ABox

Using the MiDeLoRa toolkit, we can define provers working on specialized substrate classes. We already mentioned in 5.3.1 that MiDeLoRa offers so-called *spatial ABoxes*. Then there is no longer a need for a hybrid map representation, since ABox individuals are also instances of spatial datatypes (like SBox nodes). From the point of view of a *standard* DL prover in MiDeLoRa, the spatial aspects of these nodes are invisible. However, dedicated "spatial" MiDeLoRa provers or query answering procedures (implementations of spatial query atoms) can be defined which access these spatial aspects of the nodes.

But also with a standard DL a spatial ABox can offer some benefits. Consider the $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ ABox we have generated in Section 5.5.1: RCC role assertion must be computed and added as well as closing assertions for the RCC roles. These closing assertions force the RacerPro tableau prover into a local closed domain reasoning w.r.t. the RCC roles in the ABox, as illustrated by the example evaluation of the instance retrieval query on that ABox. A MiDeLoRa prover working on a spatial ABox can, in principle, do better here, for the following reasons:

1. With a spatial ABox, the RCC role assertions must not be precomputed and added as assertions at all. They can be computed by means of inspection methods and materialized on the fly *if needed* during the tableau proof. These materialized "virtual" assertions are then treated like ordinary ABox assertions.[8] Thus, there is no need to explicitly store an $|V|^2$ number of RCC role assertions in the ABox, they are "intrinsically represented".

2. As explained, the $i : (\leq\ R\ n) \sqcap (\geq\ R\ n)$ number assertions force the tableaux prover to reuse existing ABox individuals when existential (successor generating) concepts are expanded that use an RCC role. However, this is a *two-step process in the* $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ *tableau*

[8] Note that the computation and materialization of such virtual assertion is an inference / prover-specfic task which requires meta knowledge regarding the inference problem at hand.

calculus, since first a fresh node satisfying the existential concept is created. Then, later on in the tableaux expansion process, it is found that this fresh node contradicts the $(\leq\ R\ n)$ assertion. Thus, the so-called *merge rule* identifies and merges the superfluous successors with an already existing R successor (mentioned in a role assertion).

It is obvious that this behavior of the tableaux prover could also be achieved in a more direct way if the generating rules were modified in such a way that before a fresh successor is generated for an RCC role, it is first tried to *reuse an existing successor* (however, the generating rules become non-deterministic with that modification). The tableaux rules of MiDeLoRa can be parameterized to work in such a way.

3. Moreover, specialized DLs can be implemented for this scenario, e.g. the already mentioned $\mathcal{ALCI}_{\mathcal{RCC}}$ prover [45, 35].

5.5.6 Representation Option 4 – Use an ABox + RCC Substrate

Finally, we can discuss a fourth option. The primary motivation for this option is to make some spatial functionality available to other users of the RacerPro system. Thus, in order to offer a comparable spatio-thematic query answering functionality to other users of the RacerPro system without having to add the whole SBox functionality to RacerPro (spatial datatypes and spatial indexes, etc.), we devise yet another kind of substrate, the *RCC substrate*, which captures the semantics of the RCC relations by exploiting techniques from *qualitative spatial reasoning.*

Users of RacerPro can associate an ABox $\mathcal{A}$ with an RCC substrate $\mathcal{RCC}$ by means of a hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ and query this hybrid substrate with nRQL *+ RCC query atoms* (see below). Note that here, unlike for the map substrate, now the ABox is again S_1 and the "primary" substrate (since the RCC substrate is an "add on" from the perspective of the RacerPro user). This technique is, on the one hand, more expressive, since it can also deal with vaguely given, under- or even unspecified RCC relations in an $\mathcal{RCC}$ substrate. On the other hand, we have the same problems as with representation Option 1, since the geometry of a map could not be preserved.

Let us describe the RCC substrate. The RCC substrate is basically an RCC network consistency checker which can decide *(relational) consistency of RCC networks* and *entailment of RCC relations* resp. RCC ground query atoms:

Definition 5 An RCC substrate $\mathcal{RCC}$ is a substrate such that V is a set of RCC nodes with $\mathcal{L}_{\mathcal{V}} = \emptyset$, and $\mathcal{L}_{\mathcal{E}} = 2^{\{EQ,DC,EC,PO,TPP,TPPI,NTPP,NTPPI\}}$. ■

The RCC base relations have already been discussed. An edge label represents a *disjunction of RCC base relations*, representing coarser or even unknown knowledge regarding the spatial relation (in case the set is not a singleton). Disjunctions of base relations are thus RCC relations as well. The *properties* of the RCC relations are captured by the so-called JEPD property (see Page 60) as well as the so-called *RCC composition table*. This table is used for solving the following basic inference problem: **Given: RCC relations** $R(a,b)$ **and** $S(b,c)$**. Question: Which relation** T **holds between** a **and** c**?** The table thus lists, at column for base relation R and row for base relation S, the RCC relation T. In general, T will not be a base relation, but a set, resp. a disjunctive RCC relation $\{T_1, \ldots T_n\}$. The RCC table is given as a set $\mathcal{RCC}_{\mathcal{T}}$ of sentences of the form $\{R \circ S = \{T_1, \ldots, T_n\}, \ldots\}$.

An RCC network resp. RCC substrate $\mathcal{RCC}$ *containing only base relations* can be viewed as a set of FOPL ground atoms. Such a network resp. RCC substrate is said to be *relationally consistent* iff $\mathcal{RCC}'$ is satisfiable: $\mathcal{RCC}' =\ \mathcal{RCC}\ \cup\ \{\forall x.EQ(x,x)\} \cup$
$\{\forall x,y,z.R(x,y) \wedge S(y,z) \rightarrow T_1(x,z) \vee \cdots \vee T_n(x,z)\ \mid$
$R \circ S = \{T_1, \ldots, T_n\} \in \mathcal{RCC}_{\mathcal{T}}\} \cup$
$\{\forall x,y. \bigvee_{R \in \mathrm{RCC}} R(x,y)\}\ \cup\ \{\forall x,y. \bigvee_{R,S \in \mathrm{RCC}, R \neq S} R(x,y) \wedge \neg S(x,y)\}$

For example, the network $\mathcal{RCC} = \{NTPP(a,b), DC(b,c), PO(a,c)\}$ is inconsistent, because if a is contained in b (atom $NTPPI(a,b)$), and b is disconnected from c (atom $DC(b,c)$), then a must be disconnected from c as well. The *RCC8 composition table* contains the axiom $NTPP \circ DC = \{DC\}$. Thus, $\mathcal{RCC}' \models DC(a,c)$, which contradicts $PO(a,c)$, due to the JEPD property.

Entailment of RCC relations or RCC ground query atoms can be *reduced to inconsistency checking* as follows:

$\mathcal{RCC}' \models R(a,b)$ **iff** $\mathcal{S} \cup (\{EQ, DC, EC, PO, TPP, TPPI, NTPP, NTPPI\} \setminus R)$ **is unsatisfiable.**

A (general) RCC network is relationally consistent iff at least one of its *configurations* is relationally consistent.

A *configuration of an RCC network* is obtained by choosing (and adding) one disjunct / base relation out of every non-base relation in that network. Thus, a configuration contains only base relations. For example, consider $\mathcal{RCC} = \{NTPP(a,b), DC(b,c)\}$. We have $\mathcal{RCC}' \models DC(a,c)$, since $\mathcal{RCC}' \cup \{(EQ \vee EC \vee PO \vee TPP \vee TPPI \vee NTPP \vee NTPPI)(a,c)\}$ is not relationally consistent, because none of its *configurations* $\mathcal{RCC}' \cup \{EQ(a,c)\}$ $\ldots$ $\mathcal{RCC}' \cup \{NTPPI(a,c)\}$ is relationally consistent.

Since the RCC substrate defines a notion of logical entailment, the semantics of the RCC relations will be correctly captured for query answering.

Consider the hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ with

$$\mathcal{A} = \{hamburg : german\_city, paris : french\_city, fr : country, ger : country\}$$
and
$$\mathcal{RCC} = \{NTPP(*hamburg, *ger), EC(*ger, *fr), NTPP(*paris, *fr)\}$$
and with the obvious (trivial) mapping $*$
$$* = \{(hamburg, *hamburg), (paris, *paris), (fr, *fr), (ger, *ger)\}.$$

Then, the query

$$ans(?city1, ?city2) \leftarrow city(?city1), city(?city2), DC(? * city1, ? * city2)$$

correctly returns $?city1 = hamburg, ?city2 = paris$, and vice versa, even though $DC(*paris, *hamburg)$ is not explicitly present in $\mathcal{RCC}$. Thus, unlike the $\models$ relation for the SBox which only requires spatial model checking by means of inspections methods, "spatial inference" by means of RCC constraint reasoning is required for query answering on the RCC substrate.

5.5.7 OWL and the RCC Substrate

RacerPro is also an OWL reasoner, therefore we have made the RCC substrate services available to OWL users as well. The idea is that OWL object properties can be declared as *synonyms* of RCC relations, e.g., the declaration

```
(rcc-synonym |http://...geo.owl#contains| (:ntppi :tppi))
```

declares the OWL object property `...#contains` as a synonym for the RCC8 relation $\{NTPPI, TPPI\}$. If an OWL document containing an RCC synonym is read in, the associated RCC substrate is created automatically, and corresponding RCC edges are inserted for OWL/RDF(S) fillers of RCC synonym object properties. RacerPro is the first OWL repository which offers some non trivial spatio-thematic query answering facilities.

5.5.8 Case Study Conclusion

The client code of the DLMAPS system is designed to work on (hybrid) substrates of various classes, it is thus a truly multi-dimensional system. A lot of effort has been put into abstraction layers. We have demonstrated that our architectural framework accounts very well for the identified problems **P1**–**P7.** Not a single line of code must be changed if the DLMAPS system uses a different substrate representation for the DISK. We claim that the achieved flexibility of the framework has been sufficiently demonstrated by means of that case study.

Moreover, by means of query rewriting and expansion, the different available representation options can be made transparent to the users. Ideally, users can abstract from the details of the DISK representation. Users should not need to known how and where a special aspect of the DISK is physically represented (in the ABox or SBox) in order to be able to formulate queries. The exploited query expansion and rewriting procedures are currently hard-coded, though.

The DLMAPS download page can be found under `http://www.sts.tu-harburg.de/~mi.wessel/dlmaps/dlmaps.html`.

5.6 SuQL- The Substrate Query Language Framework

In the following we describe the core design principles underlying the generic substrate query language SuQL, its instantiations resp. specializations (nRQL, SnRQL, ...), as well as the features and core optimizations found in the query answering engine.

Some ideas of the SuQL framework have already been sketched, as well as some examples for queries in abstract Horn-logic syntax been given. In the following, we will use the concrete syntax of the query language framework in order to make it "more concrete" and stress the fact that this is implemented and working code.[9] The query

$$ans(?x, ?y) \leftarrow woman(?x), has\_child(?x, ?y)$$

takes the following form in concrete syntax:

```
(retrieve (?x ?y) (and (?x woman) (?x ?y has-child))).
```

The expression `(?x ?y)` is called the *head,* and `(and (?x woman) (?x ?y has-child))` the *body* of the query. SuQL only offers so-called *distinguished* or *must-bind variables.*

SuQL offers *substrate specific* unary and binary *query atoms* (whose concrete syntax may be defined accordingly), from which *complex queries* can be constructed using the body constructors `and, or, neg` and `project-to`; `neg` corresponds to "\", and `project-to` to "π", as already shown and briefly discussed in the example queries in Section 5.5.2 (moreover, `union` is accepted for `or` as well, and `pi` for `project-to`). These body constructors can be combined / nested in an arbitrary (orthogonal) way. This is why we call SuQL an orthogonal language.

If we assume that `(?x woman)` is a *concept query atom*, – a specialized unary query atom for substrates of class *ABox* –, and `(?x ?y has-child)` is a *role query atom* – a specialized binary query atom for for substrates of class *ABox* –, then, if posed to a substrate of type ABox, the query returns all mother-child pairs from that ABox.

SuQL has the following peculiarities which we want to discuss briefly before syntax and semantics is specified, since this discussion will greatly enhance the comprehensibility of the algebraic specification of the semantics:

Variables and individuals can be used in query atoms. The variables range over V, the nodes of the query substrate (this is called the *active domain semantics*). Variables are bound to nodes which *satisfy* the query – a variable binding satisfies a query iff the ground query – that is obtained by replacing all variables with their bindings – is logically entailed by the substrate. For example, the atom $P(x)$ is satisfied in substrate S if $x = i$, $i \in V$ and $S \models P(x)_{x \leftarrow i}$ resp. $S \models P()$. Thus, a variable is only bound to a substrate node iff it can be proven that this binding holds in *all* models of the substrate.

Returning to our example query body `(and (?x woman) (?x ?y has-child))`, `?x` is only bound to those individuals which are instances of the concept `woman` having a *known* child `?y` in *all* models of the KB.

SuQL offers 4 kinds of variables: Variables which begin with `$?` are called *injective variables*, in contrast to the ordinary non-injective variables beginning with `?`. For an injective variable it is required that it can only be bound to a substrate node that is not already bound by another injective variable (thus, the function mapping injective variables to substrate nodes must be injective; in the past, we have called this a "UNA for variables"). Given the availability of (negated) equality atoms in SuQL we can neglect injective variables in the following semantics definition. By means of query rewriting, such atoms can be added automatically. Moreover, if a hybrid substrate is queried, then variables for S_2 must start with either `?*` or `$?*`, as discussed.

Negation as Failure (NAF) Operator: The `neg` operator implements a *Negation as Failure Semantics (NAF).* For example, `(neg (?x woman))` returns all substrate nodes for which it *cannot be proven* that they are instances of `woman`.

Thus, `(neg (?x woman))` returns the complement set of `(?x woman)` (w.r.t. to V, the set of all substrate nodes). If a binary query atom is NAF negated, e.g. `(neg (?x ?y has-child))`, then the complement is two-dimensional. Thus, all pairs of individuals are returned which are not in the `has-child` relation.

[9] The prefix Lisp syntax is as clean and readable and mathematically profound as the mathematical syntax.

Let us define the *extension* of a unary (binary) query atom $P(?x)$ ($Q(?x,?y)$) as the query answer of the query $ans(?x) \leftarrow P(?x)$ (resp. $ans(?x,?y) \leftarrow Q(?x,?y)$, and denote that extension as $P(?x)^{\mathcal{E}}$ (resp. $Q(?x,?y)^{\mathcal{E}}$). It is obvious that the following equalities must hold, for any substrate S with nodes V:

$$\begin{aligned} V &= P(?x)^{\mathcal{E}} \cup (\backslash P(?x))^{\mathcal{E}} \\ V \times V = V^2 &= Q(?x,?y)^{\mathcal{E}} \cup (\backslash Q(?x,?y)).^{\mathcal{E}} \end{aligned}$$

However, things become more tricky if individuals are used in query atoms. What should be the extension of such an atom? For reasons of orthogonality we replace individuals (from V) in atoms with representative variables and add equality conjuncts, e.g., the atom $Q(i,?y)$ with $i \in V$ is rewritten into the conjunction $Q(?x_i,?y), ?x_i = i$. This ensures that the extensions of the atoms always have the same dimensionality, no matter whether the atom references 0, 1 or 2 individuals, and that the above given equalities hold. This enables the algebraic specification of the semantics given subsequently. For the extension of the atom $\backslash Q(?x_i,?y)$ we must thus necessarily get:

$$\begin{aligned} (\backslash Q(i,?y))^{\mathcal{E}} &= V^2 \setminus Q(i,?y)^{\mathcal{E}} \\ &= V^2 \setminus (Q(?x_i,?y), ?x_i = i)^{\mathcal{E}} \\ &= V^2 \setminus (Q(?x_i,?y)^{\mathcal{E}} \cap (?x_i = i)^{\mathcal{E}}) \\ &= V^2 \setminus (Q(?x_i,?y)^{\mathcal{E}} \cap \{\, (x,y) \mid\ x = i, y \in V \,\}\ ) \\ &= \backslash Q(?x_i,?y)^{\mathcal{E}} \cup \{\, (x,y) \mid\ x \neq i, y \in V \,\}. \end{aligned}$$

This is surprising at a first glance but unavoidable if DeMorgan's Law shall hold for SuQL bodies. The validity of DeMorgan's Law is crucial for the definition of the *Disjunctive Normal Form (DNF)* for queries, which in turn is indispensable for the cost-based query optimizer (see below).

Let us consider the ABox query language case again. We would like to emphasize that `(?x (not woman))` has a different semantics from `(neg (?x woman))`, since the former returns the individuals for which the DL system *can prove* that they are *not instances* of woman, whereas the latter returns all instances for which the DL system *cannot prove* that they are *instances* of woman. Also note that `neg` and `not` are equivalent on substrates which employ the CWA (e.g., the SBox).

Different Notions of Equality are Available: Equality query atoms are added automatically as soon as individuals are referenced, as just shown. The equality atoms are also explicitly available for query formulation.

Equality atoms can either use syntactic or semantic equality predicates: "$=_{syn}$" or "$=_{sem}$"; these notions coincide if the UNA is used.[10]

Let us illustrate the different equality predicates using the example ABox

$$\mathcal{A} = \{(i,j) : R, (i,k) : R, j : C, i : (\leq_1 R)\}.$$

Obviously, $\mathcal{A} \models j =_{sem} k$, but $\mathcal{A} \not\models j =_{syn} k$ and thus $\mathcal{A} \models j \neq_{syn} k$ (and also $\mathcal{A} \models \backslash(j =_{syn} k)$). For $C(i)^{\mathcal{E}} = (C(x_i), x_i = i)^{\mathcal{E}}$ we thus get either $\{i\}$ or $\{i,j\}$, depending on whether "$=_{syn}$" or "$=_{sem}$" is used. Note that the answer $\{i,j\}$ is *convenient* and could not be delivered if the discussed rewriting would not take place. With "$=_{syn}$" we would get $(\backslash C(i))^{\mathcal{E}} = \{j\}$, which might be misleading. Of course, $\emptyset$ returned in the "$=_{sem}$" case.

Alternatively, also the *domain of the variables* can be changed: Instead of requiring that a variable is bound to a substrate node (resp. an ABox individual), we can also require that a variable is bound to a *"$=_{sem}$" equivalence class*, or, more precisely, to *one representative node* in that equivalence class.

The Body Projection Operator (`project-to`): This operator is required in order to reduce the "dimensionality" of the extension of a subbody in a query body, for example, before computing the complement set of the extension of that subbody with `neg`.

Let us motivate the necessity for such an operator, again using the ABox case. Consider `(retrieve (?x) (and (?x mother) (?x ?y has-child)))`. This query returns all mothers having a *known child* in the ABox. Now, how can we query for mothers who do *not* have a *known* child? Our first attempt will be the query `(retrieve (?x) (and (?x mother) (neg (?x ?y has-child))))`. A bit of thought and recalling that `(neg (?x ?y has-child))` returns the complement set of `(?x ?y has-child)` w.r.t. the Cartesian product of all ABox individuals will reveal

[10] The predicate $=_{sem}$ is the standard equality predicate in FOPL with equality. Surprisingly, $=_{syn}$ is indeed definable by restricting $=_{sem}$, but certain encoding tricks are needed which we do not want to discuss here.

that this query doesn't solve the task. In a second attempt, we will probably try `(retrieve (?x) (neg (and (?x mother) (?x ?y has-child))))`. However, due to DeMorgan's Law this query is equivalent to `(retrieve (?x) (union (and (neg (?x mother)) (?y top)) (neg (?x ?y has-child))))` – first the union of two two-dimensional tuple sets is constructed, and then only the projection to the first element of these pairs (`?x`) is returned. Obviously, this set contains also the instances which are *not* known to be mothers, which is wrong as well. Thus, the need for the projection operator becomes apparent: `(retrieve (?x) (and (?x mother) (neg (project-to (?x) (?x ?y has-child)))))` solves the task. In principle, this corresponds to the query $ans(?x) \leftarrow mother(?x), \backslash hac(?x)$, where $hac(?x) \leftarrow has\_child(?x, ?y)$.

Head Projection Operators and Query Nesting. It is often the case that some subsequent computations based on the variable bindings of the "current" answer tuple shall be performed. For this purpose, not only individuals and variables can appear in a query head, but also so-called head projection operators. These are denoted in a functional style using lambda expressions. For example, the query

```
(retrieve (?r ((lambda (l w) (* l w)) ?l ?w))
   (and (?r rectangle) (?r ?l has-length) (?r ?w has-width)))
```

returns, for the actual bindings `?r = r1, ?l = 10, ?w = 20`, the result tuple `((?r r1) ((lambda ...) 200))`.[11]

Most importantly, also *subqueries* can be posed in lambda bodies. Thus, SuQL allows for arbitrary query nesting. With this functionality and an extensive *termination-safe* expression language for lambda bodies (which we call MiniLisp), for example, the equivalent of SQL aggregation operators (count, sum, max, min, average, ...) is available in SuQL.

For specific SuQL instantiations, syntactic sugar for certain (idiomatic) lambda expressions can be defined.

5.6.1 Syntax and Semantics

We only specify syntax and semantics for non-hybrid queries; the extension to hybrid queries is straightforward, but does not really add to this paper:

Definition 6 (Syntax of SuQL) The *head* and *body* of a SuQL query `(retrieve` *head body*`)` are defined by the following BNF grammar (note that {a|b} represents `a` or `b`):

head	:=	`(`*head_entry**`)`
object	:=	*variable* | *individual*
variable	:=	a symbol beginning with `?`
individual	:=	a symbol
head_entry	:=	*object* | *lambda*
lambda	:=	`((lambda ...)`*variable**`)`
body	:=	*atom* | `(` {`and` | `union`} *body**`)` | `(neg` *body*`)` | `(project-to (`*object**`)` *body*`)`
atom	:=	*unary_atom* | *binary_atom* | *equality_atom*
unary_atom	:=	`(`*object unary_atom_predicate*`)`
binary_atom	:=	`(`*object object binary_atom_predicate*`)`
equality_atom	:=	`(`*object object* {$=_{syn}$ | $=_{sem}$}`)`

The predicates *unary_atom_predicate* and *binary_atom_predicate* are conceived as FOPL formulas with one resp. two free variables x and y; however, the concrete syntax may offer a variable-free syntax for them.

The function $\mathsf{obs}(q)$ is defined inductively as follows: $\mathsf{obs}(unary\_atom) =_{def} \{x_1\}$ if $unary\_atom =$ `(`x_1 *unary_atom_predicate*`)`, $\mathsf{obs}(binary\_atom) =_{def} \{x_1, x_2\}$ if $binary\_atom =$ `(`x_1 x_2 Q`)` with $Q \in \{binary\_atom\_predicate, =_{syn}, =_{sem}\}$, $\mathsf{obs}(($`{ and | union | neg }` $q_1 \ldots q_m)) =_{def} \bigcup_{1 \leq i \leq m} \mathsf{obs}(q_i)$, but $\mathsf{obs}($`(project-to (`$x_1 \ldots x_m$`) ...)`$) =_{def} \{x_1 \ldots x_m\}$. Thus, obs "stops at projections". ■

[11] Note that `((lambda (l w) (* l w)) ?l ?w)` denotes an application of the anonymous function `(lambda (l w) (* l w))` to the actual bindings of `?l` and `?w` (10 and 20); the evaluation of the lambda expression yields thus 10*20=200.

Definition 7 (Semantics of SuQL) Let $S = (V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$ be a substrate, and q be a *body*.

First we must define some auxiliary operations. Let $\mathcal{T}$ be a set of n-ary tuples $\langle t_1, \ldots, t_n \rangle$ and $\langle i_1, \ldots, i_m \rangle$ be an *index vector* with $1 \leq i_j \leq n$ for all $1 \leq j \leq m$. Then we denote the set $\mathcal{T}'$ of m-ary tuples with $\mathcal{T}' =_{def} \{ \langle t_{i_1}, \ldots, t_{i_m} \rangle \mid \langle t_1, \ldots, t_n \rangle \in \mathcal{T} \} = \pi_{\langle i_1, \ldots, i_m \rangle}(\mathcal{T})$, called the *projection* of $\mathcal{T}$ to the components mentioned in the index vector $\langle i_1, \ldots, i_m \rangle$. For example, $\pi_{\langle 1,3 \rangle}\{\langle 1,2,3 \rangle, \langle 2,3,4 \rangle\} = \{\langle 1,3 \rangle, \langle 2,4 \rangle\}$.

Let $\vec{b} = \langle b_1, \ldots, b_n \rangle$ be a *bit vector* of length n, $b_i \in \{0,1\}$. Let $m \leq n$. If $\vec{b}$ is a bit vector which contains exactly m ones, and $\mathcal{B}$ is some set ("the base"), and $\mathcal{T}$ is a set of m-ary tuples, then the n-dimensional *cylindrical extension* $\mathcal{T}'$ of $\mathcal{T}$ w.r.t. $\mathcal{B}$ and $\vec{b}$ is defined as

$$\begin{aligned}\mathcal{T}' =_{def} \{ \langle i_1, \ldots, i_n \rangle \mid \quad & \langle j_1, \ldots, j_m \rangle \in \mathcal{T}, 1 \leq l \leq m, 1 \leq k \leq n \\ & \text{and } i_k = j_l \text{ if } b_k = 1 \text{ and } b_k \text{ is the } l\text{th ``1'' in } \vec{b}, \\ & \text{and } i_k \in \mathcal{B} \text{ otherwise. } \}\end{aligned}$$

and denoted by $\chi_{\mathcal{B},\langle b_1,\ldots,b_n \rangle}(\mathcal{T})$. For example,
$\chi_{\{a,b\},\langle 0,1,0,1 \rangle}(\{\langle x, y \rangle\}) = \{\langle a,x,a,y \rangle, \langle a,x,b,y \rangle, \langle b,x,a,y \rangle, \langle b,x,b,y \rangle\}$.

We denote an n-dimensional bit vector having ones at positions specified by the index set $\mathcal{I} \subseteq 1 \ldots n$ as $\vec{\mathbf{1}}_{n,\mathcal{I}}$. For example, $\vec{\mathbf{1}}_{4,\{1,3\}} = \langle 1,0,1,0 \rangle$. Moreover, with $\mathcal{ID}_{n,\mathcal{B}}$ we denote the n-dimensional identity relation over the set $\mathcal{B}$: $\mathcal{ID}_{n,\mathcal{B}} =_{def} \{ \overbrace{\langle x, \ldots, x \rangle}^{n} \mid x \in \mathcal{B} \}$.

The semantics of a query is given by the set of tuples it returns if posed to a substrate S. This set of answer tuples is called the extension of q and denoted by $q^\mathcal{E}$.

First we add the mentioned equality atoms for query atoms which reference individuals. The query body q is thus first rewritten. We define $\Theta(q)$ for query atoms *atom* with $\mathsf{obs}(atom) \cap V = \{v_1, \ldots, v_n\}$, $n \in \{1,2\}$ as

$$\Theta(atom) =_{def} (\texttt{and } atom\ (x_{v_1}\ v_1\ =)\ \ldots\ (x_{v_n}\ v_n\ =)),$$

(please note that $= \in \{=_{syn}, =_{sem}\}$, as previously discussed, and that x_{v_i} is the representative variable for v_i) and extended the definition of Θ in the obvious (inductive) way to complex query bodies as well. Moreover, Θ replaces all occurrences of individuals in the projection list of `project-to` and in the query *head* with their representative variables.

Let $q' = \Theta(q)$ be the rewritten query. So we simply declare $q^\mathcal{E} =_{def} q'^\mathcal{E}$. Let us specify $q'^\mathcal{E}$. Let $\langle x_{1,q'}, \ldots, x_{n,q'} \rangle$ be a lexicographic enumeration of $\mathsf{obs}(q')$ (so $n = |\mathsf{obs}(q')|$). Denote the ith element in this vector with $x_{i,q'}$, indicating its position in this vector.

We define $\cdot^\mathcal{E}$ inductively. We start with the query atoms:

$$\begin{aligned}(x_{i,q'}\ P)^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i\}}}(\{ < v > \mid v \in V, S \models P_{x \leftarrow v} \}\ ) \\ (x_{i,q'}\ x_{j,q'}\ Q)^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, S \models Q_{x \leftarrow u, y \leftarrow v} \}\ )\end{aligned}$$

(please note that due Θ, all unary and binary query atoms which are not equality atoms now have two variables as objects). The semantics of the equality predicates resp. atoms is fixed as follows:

For $=_{syn}$: $S \models i =_{syn} i$ and $S \not\models i =_{syn} j$.

For $=_{sem}$: $S \models i =_{sem} j$ iff for all models $\mathcal{I}$ of S ($\mathcal{I} \models S$): $i^\mathcal{I} = j^\mathcal{I}$,

Thus, we define:

$$\begin{aligned}(x_{i,q'}\ x_{j,q'}\ =_{syn})^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, \\ &\qquad \text{if } x_{i,q'} \in V, \text{ then } u = x_{i,q'}, \text{ if } x_{j,q'} \in V, \text{ then } v = x_{j,q'} \}) \\ (x_{i,q'}\ x_{j,q'}\ =_{sem})^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, S \models u =_{sem} v, \\ &\qquad \text{if } x_{i,q'} \in V, \text{ then } u = x_{i,q'}, \text{ if } x_{j,q'} \in V, \text{ then } v = x_{j,q'} \}).\end{aligned}$$

We extend the definition of $\cdot^\mathcal{E}$ inductively for complex (sub)bodies in q' as follows:

$$\begin{aligned}(\texttt{and } q'_1 \ldots q'_i)^\mathcal{E} &=_{def} \textstyle\bigcap_{1 \leq j \leq i} q'^\mathcal{E}_j \\ (\texttt{union } q'_1 \ldots q'_i)^\mathcal{E} &=_{def} \textstyle\bigcup_{1 \leq j \leq i} q'^\mathcal{E}_j \\ (\texttt{neg } q'_1)^\mathcal{E} &=_{def} V^n \setminus q'^\mathcal{E}_1 \\ (\texttt{project-to } (x_{i_1,q'} \ldots x_{i_k,q'})\ q'_1)^\mathcal{E} &=_{def} \pi_{\langle i_1,\ldots,i_k \rangle}(q'^\mathcal{E}_1)\end{aligned}$$

To get the final answer of a query, the *head* has to be considered, for a final projection. Moreover, the `lambda`s must be applied to the current bindings, and the computed result included in the answer tuple. For the sake of brevity we only consider heads without lambdas. Thus, the result of `(retrieve` *head body*`)` is simply given as

$$(\texttt{retrieve}\ head\ body)^{\mathcal{E}} =_{def} (\texttt{project-to}\ \Theta(head)\ \Theta(body))^{\mathcal{E}}.$$

■

5.6.2 nRQL is a Special SuQL

As already mentioned, nRQL is a specialized SuQL. As such, it offers dedicated query atoms for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, especially also atoms concerning the *concrete domain and concrete domain constraint checking* which is a unique RacerPro feature, as well as special syntactic sugar for certain idiomatic lambda operators, again primarily for the concrete domain as well as for some RacerPro specific OWL peculiarities.

The nRQL atoms are: (unary) *concept query atoms*, e.g. `(?x (some has-child human))`; (binary) *role query atoms,* e.g. `(?x ?y has-child)`, and (binary) *constraint query atoms.* All but the constraint query atoms have been discussed already. Here is a nRQL example query utilizing all three kinds of atoms:

```
(retrieve (?x)
      (and (?x (and woman (min age 40))) (?x ?y has-child)
           (?y ?y (constraint (has-father age) (has-mother age)
                              (<= (+ age-2 8) age-1)))))
```

This query returns thus instances of the concept `women` which are older than 40 and which have children whose fathers are at least 8 years older than their mothers. Note that `(has-father age)` denotes a role chain ended by a so-called concrete domain attribute, a kind of "path expression": starting from the individual bound to `?y` (the child), we retrieve "the value" of the concrete domain attribute `age` of the individual which is the filler of the `has-father` role (feature) of this individual. In a similar way, the age of the mother of `?y` is retrieved. These concrete domain values are then used as actual arguments to check whether the predicate `(<= (+ age-2 8) age-1)` holds for them; `age-2` refers to `(has-mother age)`, and `age-1` refers to `(has-father age)`.[12] However, these "values" are in fact variables in a concrete domain constraint network (which can be left unspecified, i.e., no told value must exist). RacerPro offers this kind of concrete domain constraint checking.

Instead of only simply role name, also more general *role terms* are admissible in role and constraint query atoms; a role term is an element in the set of role names closed under the operators {`not`, `inv`}. Thus, nRQL offers not only NAF negated roles, but also *classical negated roles.* For example, the extension of `(?x ?y (not has-father))` contains those pairs resp. bindings for `?x, ?y` for which RacerPro *can prove* that the individual bound to `?x` cannot have the individual bound to `?y` as a father. If the role `has-father` was defined as having the concept `male` as a range, then at least all pairs of individuals in which `?y` is bound to a `female` person are returned, given `male` and `female` can be proven disjoint. Moreover, we have the NAF negation, of course, so `(neg (?x ?y has-father))` simply returns the two-dimensional complement of all entailed `has-father` relationships. Even `(neg (?x ?y (not has-father)))` can be used.

Moreover, the concept terms resp. expressions have been extended to support OWL querying better; the main point here is to provide more transparency and a more user friendly syntax, given that the internal representation of *OWL datatype properties* in RacerPro which are mapped to the concrete domain is rather complex (but provides much higher expressivity than alternative approaches).

Note that the presence of role chains in constraint query atoms requires some additional rewriting from Θ. Basically, role chains in constraint query atoms are broken up into conjunctions of simple role query atoms as well the constraint query atom referencing only the concrete domain attributes.

[12] Note that the suffixes `-1, -2` have been added to the `age` attribute in order to differentiate the two values (the mechanism is not needed in case the two chains are ended by different attributes).

Atoms of the syntax `(same-as ?x i)` are used to denote equality atoms. Whereas older nRQL version have used $=_{syn}$ for `same-as`, it is now possible to use $=_{sem}$ as well. This mainly concerns the equality atoms which are automatically added by Θ; moreover, the so-called special "nRQL equal role" was (and still is) available which is interpreted as $=_{sem}$ as well.

Regarding the head projection operators, nRQL offers support for retrieving so-called *told values* from the concrete domain as well as support for retrieving fillers of OWL (annotation) datatype properties. The problem here is that variables can only be bound to ABox individuals resp. nodes in V. Since XML datatype values (being fillers of OWL datatype properties) or concrete domain values (e.g., Integers) are not represented as ABox individuals, no bindings can be computed. However, *retrieval conditions* can be specified by means of concept and constraint query atoms. Thus, in order to actually retrieve these datatype fillers, special head projection operators are offered; returning to our previous example query, we can simply use the head `(?x (told-value (age ?x)))` to also get the told fillers of the `age` attribute (given such told fillers exist); the head operator is conceptually just syntactic sugar for a more complex `lambda` entry. For OWL, special operators such as `datatype-fillers, annotation-fillers` etc. are offered in addition. Please refer to [46] for more details.

Given the generic semantics definition, it should be clear how the semantics of the dedicated nRQL atoms can be defined. Basically, we just need to define $S \models P_{x \leftarrow v}$ as well as $S \models Q_{x \leftarrow u, y \leftarrow v}$; note that S is now an ABox $\mathcal{A}$. However, this is easy using the standard-translation Φ of DL into FOPL [39]; e.g., for a concept query atom predicate $P = C$ this boils down to ordinary instance checking resp. instance retrieval query: $\mathcal{A} \models \Phi(C)_{x \leftarrow i}$ iff $\mathcal{A} \models i : C$ iff $\mathcal{A} \cup \{i : \neg C\}$ is unsatisfiable (basically, just the RacerPro API functions concept_instances or individual_instance? needs to be called), and for positive roles R in role atoms we get $\mathcal{A} \models \Phi(R)_{x \leftarrow i, y \leftarrow j}$ iff $\mathcal{A} \models (i, j) : R$ iff $\mathcal{A} \cup \{i : \forall R.M, j : \neg M\}$ is unsatisfiable, for some fresh concept name M (again, there are standard API functions: role_fillers and individuals_related?). However, for negated roles, we need to perform an ABox satisfiability (consistency) check: $\mathcal{A} \models \Phi(\neg R)_{x \leftarrow i, y \leftarrow j}$ iff $\mathcal{A} \cup \{(i, j) : R\}$ is unsatisfiable. These "reduction tricks" are well-known [47]. RacerPro specific API functions are also called for constraint query atoms.

5.6.3 Concrete SuQL Instantiations for the DLMAPS System

We have discussed four representation options in the DLMAPS system. Although the principle ideas have been laid out, we briefly want to discuss the resulting spatio-thematic query languages in the SuQL framework for the DLMAPS system. Which spatio-thematic QL is now applicable for the different representation options (1–4) in the DLMAPS system? Again we like to stress that the user must in principle not be bothered with the representation details, since a surface end user syntax can be offered by means of query rewriting and expansion:

Option 1: We can only use plain nRQL, as explained.

Option 2: The resulting hybrid QL is called SnRQL. It provides the following spatial atoms in addition to nRQL (note that is does not really add to the message of this text to define these here formally). The extensions of the atoms are computed on the fly by means of inspections methods. As argued, this can be understood as a kind of (spatial) model checking. The following additional spatial atoms are currently offered:

RCC atoms: Atoms such as `(?x ?y (:tppi :ntppi))`; `(:tppi :ntppi)` denotes the disjunctive RCC relation $\{TPPI, NTPPI\}$. A rich set of common sense natural language spatial prepositions such as `:contains, :adjacent, :crosses, :overlaps, :flows-in` is available. The Θ function rewrites these into (the closest possible) RCC relation.

Distance Atoms: `(?x ?y (:inside-distance <min> <max>))`, where `<min>, <max` specifies an interval $[min; max]$; `NIL` can be used for 0 resp. ∞ (this applies to the subsequent interval specifications as well). For example, the extension of `(i ?x (:instance-distance nil 100))` consists of all SBox objects which are *not further away than 100 meters from i*. Either the shortest distance or the distance of the centroids of these objects can be used for distance computation.

Epsilon Atoms: `(?x ?y (:inside-epsilon <min> <max>))`. With that atom, all objects `?y` are retrieved, such that `?y` is contained within the *buffer zone* of size $[min; max]$ around `?x`. This buffer zone consists of all points (x, y) whose *shortest distance to the fringe of (the individual bound to)* `?x` is contained within $[min; max]$.

Geometric Attribute Atoms: Atoms regarding geometric attributes, e.g. length and area: The extension of `(?x (:area 100 1000))` consists of all nodes of SDT type polygon in V whose area is in $[100; 1000]$. Also `:length` is understood, but for SDTs of type line or chain.

Moreover, simple type checking atoms such as `(?x :is-polygon)`, `(?x :is-line)` etc. are available (these are needed in order to guard the application of certain spatial operators).

Let us give one concrete SnRQL query which selects an appropriate home for a millionaire:

```
(retrieve (?villa ?living-area  ?golf-club ?church)
  (and (?*living-area (and living-area
                       (or (all classification first-class-area)
                           (string= name "Beverley Hills"))))
       (?living-area ?villa :contains)
       (?*villa (and villa
                   (all status for-sale) (> has-price 10000000)
                   (some has-comfort swimming-pool)))
       (?church ?living-area (:inside-epsilon nil 200))
       (?living-area ?golf-club :adjacent)
       (?*golf-club (and golf-club (all members millionaire)))))
```

Option 3: In principle like SnRQL, but no hybridness is needed. Moreover, the MiDeLoRa prover currently does not offer concrete domains. Thus, the ABox query language part is reduced to concept and role query atoms.

Option 4: The resulting hybrid QL is called nRQL *+ RCC atoms*. This language can only offer RCC atoms in addition to nRQL, since the geometry of the map is not represented. These RCC atoms have the same syntax as the SnRQL RCC atoms, but their implementation is of course different (since RCC constraint checking is required in the latter case, and geometric computations in the former case). Spatial natural language prepositions are understood as well. Note that this language is available to all RacerPro users and also works for OWL, as described.

5.7 The SuQL Query Answering Engine

Let us briefly describe some core features of the SuQL engine which make it unique:

- The engine supports *full life-cycle management for queries:* A query is first parsed and compiled (prepared). It is available as a Clos object then. The query can then be executed, it will become active. An active query can be suspended or aborted, and will eventually terminate. After this, the query is still not gone. It can be re-prepared, re-executed etc. as long as it is not explicitly deleted.

- The runtime resources acquired by the engine are configurable: size of thread pool, maximum bound on the number of answer tuples setable, timeout setable, tuple permutations can be excluded, etc.

- Another important feature of the engine is the *built-in query optimizer* which uses syntactic as well as semantic optimization techniques (see below).

- In principle, there can be more than one query running at a time. The engine *offers concurrent querying.* When a query is executed, a *thread* from a *thread pool* is acquired and put to work. The engine can process up to a few hundred queries simultaneously. Especially with today's available multi-core processors this is an attractive feature.[13]

- The queries can be compiled into native machine code by means of the COMMON LISP compiler if maximum performance is needed.

- The engine offers *different querying modes:* basically, a synchronous *set-at-a-time mode* as well as an asynchronous *tuple-at-a-time mode.*

  In the *set-at-a-time mode*, a call to a querying function such as `retrieve` works synchronously (blocking). The client has to wait, the whole answer set is delivered in a bunch. However, many client applications prefer an asynchronous (non-blocking) API: the *tuple-at-a-time mode* allows for an incremental, cursor-based retrieval of the answer set *tuple by tuple.* Thus, a call to `retrieve` will return immediately with a so-called *query identifier.* This identifier, say `:query-123`, is then used as argument to functions such as `(get-next-tuple :query-123).` Functions like `(get-next-n-tuples 10 :query-123)` are provided as well.

  Moreover, the tuple-at-a-time mode works either *lazy or eager*: In the lazy mode, the next tuples will not be computed before requested by the client, unlike the eager mode, which precomputes the next tuple(s) and puts them into a queue for future requests. **Thus, the lazy mode only acquires as many computational resources as needed for answering the users request.** This is a very important feature, especially if the computational complexity for query answering is high.

Especially for ABox query languages such as nRQL or the MIDELORA ABox query language (which is like nRQL, but without constraint query atoms, OWL support etc.), the following features are of importance; please note that nRQL is *integrated* into RACERPRO, but still *layered on top of* RACERPRO, thus, the official API functions are used:

- Most DL system today only offer a set-based API, e.g. API functions such as concept_instances work synchronously and blocking. They can only return the complete set of answers. However, for an *incremental, cursor-based* ABox query language such as nRQL, also incremental, cursor-based API functions should be present. Otherwise there is an *impedance mismatch:* Consider an ABox query such as `(retrieve (?x) (?x woman))`. If this query is executed in incremental mode, an API call `(concept-instances woman)` is generated (if the answer is not already cached). Even if only one instance is requested by means of `get-next-tuple`, the DL system has already computed and returned *all* instances. For very complex KBs, this might be a bad idea (especially if the client requests only a very few tuples). Alternatively, the query answering engine could consider the candidate individuals element-wise and use only test functions, e.g. single individual_instance? tests. Unfortunately, this likely results in a bad performance (since the prover state resp. context is lost during the individual calls; of course, there might be caching, but that does not really solve the overall problem).

  Please note that the impedance mismatch cannot be solved by simply augmenting the set-based API functions, e.g. concept_instances, with an additional (optional) argument "number of answers", since also the *cursor position* and ideally also the internal *state of the prover* (e.g., its caches, the already performed deterministic parts of the tableaux expansions, etc.) should be preserved during the calls. The MIDELORA DL system therefore offers a continuation-based API which naturally (i.e., without caching) allows to keep the current state.

- In order to achieve maximal parallelism, all inference answers from RACERPRO are cached. The implies that calls to RACERPRO can be reduced (due to cache hits), and thus, locking of RACERPRO (and therefore blocking situations which occur if two or more queries running in parallel both require the reasoner at the same time) is minimized. These cache and index structures are automatically invalidated if changes to the queried KB occur. If a change occurs during incremental query execution, a *"KB has changed" warning token* can

[13] Thanks to Kay Hidde for pointing that out.

be delivered. Such an "outdate" query can be executed again (due to the query life cycle management).

- The *degree of completeness of query answering in* nRQL *is configurable:* On complex KBs, complete ABox retrieval might be impossible, due to the computational complexity. This is a pity if at least some subset of the overall answer could be delivered, especially in the incremental modes. Thus, nRQL offers various degrees of completeness for query answering. This is a reasonable functionality in many application scenarios, and its availability does not state that the authors think that completeness is not important.

- *Having incomplete modes available gives* nRQL *the ability to distinguish between cheap and expensive tuples.* It is possible to advise nRQL first to deliver a set of *cheap tuples*, yielding an incomplete answer ("phase one"), and then to turn on RacerPro's ABox retrieval functions to deliver the remaining *expensive tuples* ("phase two"); these tuples are potentially called "expensive" since inference is required to compute them. We have called this processing pattern the *two-phase query processing mode.* Again, nRQL can be advised to deliver a *warning token* before phase two starts, informing the client that computation of the remaining tuples will eventually be expensive. The client can then choose to retrieve these additional tuples or not (e.g., abort the query).

5.7.1 Mission Critical Optimizations

The SuQL engine exploits two optimization techniques: a cost-based syntactic optimizer [48] and a semantics-based optimization, the so-called *query repository* [48, 46], which we do not describe in detail here. But optimizations are generic and constitute to the basic functionality of the engine.

The cost-based optimizer first transforms the body of the query into *Disjunctive Normal Form (DNF).* In the DNF, each disjunct of the DNF is either a single atom or a conjunctive query. Each conjunctive query is optimized individually. To do so, the optimizer generates a potential $n!$ number of possible *execution plans*, if n conjuncts are present. An *execution plan* of a conjunctive query determines the oder of sequence in which the atoms are about to be evaluated. In order to determine the "costs" of an atom in a plan, a method `get-score` is called, which can be overwritten for specialized atoms. A plan is thus generated successively, and given a plan $(a_1, \ldots, a_n)$, an atom a_{n+1} is selected and added next to the plan which yields the maximal score: $(a_1, \ldots, a_n, a_{n+1})$. In such a way, a heuristic search procedure using *beam search (of sufficient breadth)* searches for a "good" plan (if n gets big, not all of the $n!$ plans can be considered and scored).

The standard implementation of `get-score` simply considers the *role* of the atom in the currently evaluated plan and weights the atom accordingly. An atom can either play the role of a *tester* or of a *generator*, depending on whether the variable(s) referenced in the atom will be already bound at execution time or not.

The standard `get-score` implementation simply prefers *testers over generators.* Moreover, *successor or predecessor generators* are preferred over successor generators. For example, consider the query `(and (?x ?y R) (?y D) (?x C))`. Using the standard `get-score` implementation, of that $3! = 6$ plans, the plans (a)=`(?x C), (?x ?y R), (?y D)` are (b)= `(?y D), (?x ?y R), (?x D)` get the highest overall score. This optimization strategy is reasonable if one assumes that the average number of `R`-successors or of `R`-predecessors of an individual is small compared to the number of `C` and/or `D` instances. Thus, the "navigational" approach for computing bindings is preferable over, e.g., the cross product generation (e.g., in the plan `(?x C), (?y D), (?x ?y R)`).

The `get-score` method is refined for nRQL. Here, also *ABox statistics* are taken into account. This enables a preference selection: plan (a) will be preferred iff $|(?x\ C)^{\mathcal{E}}| \leq |(?y\ D)^{\mathcal{E}}|$ (and (b) otherwise). Unfortunately, this information is not always available, and in many cases, one has to rely on *told information* only. Nevertheless the refined valuation has been proven valuable.

We have performed an evaluation of the effectiveness of this optimization technique, using the *Lehigh University Benchmark (LUBM)* and the LUBM Query No. 9 (**Q9** in the following) [49]. For instance, **Q9** is the following query:

```
(retrieve (?x ?y ?z)
```

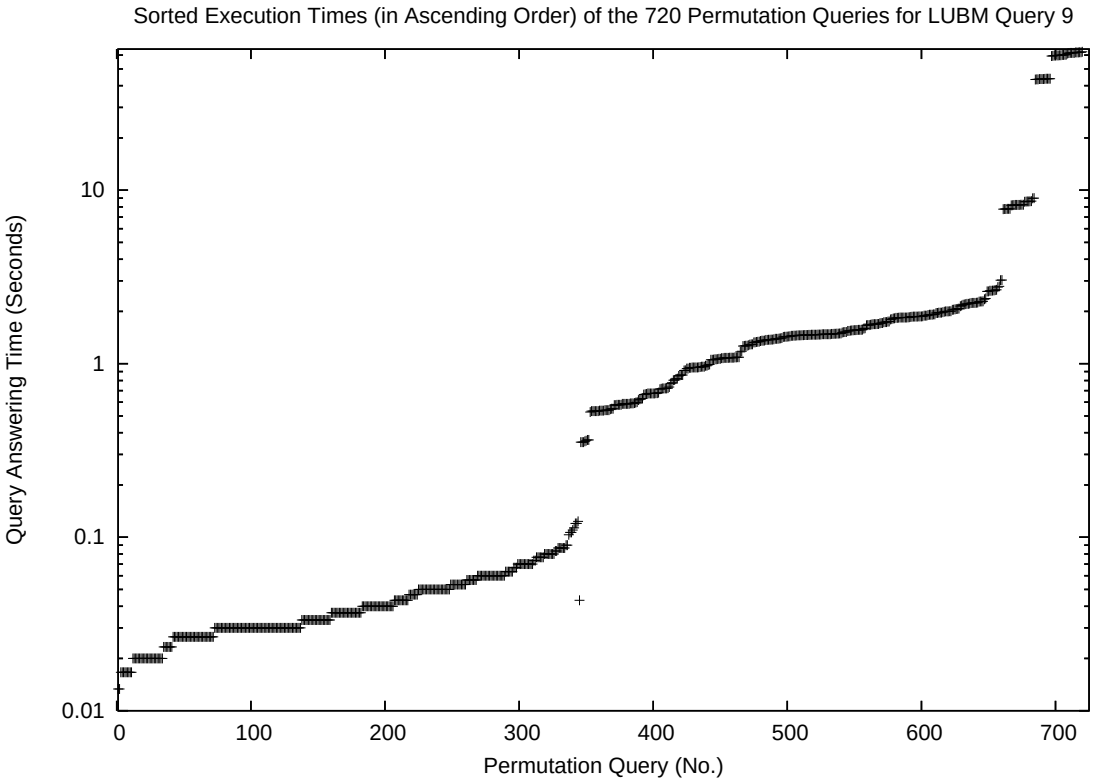

Figure 5.5: Execution Times of the 720 Permutation Queries for **Q9**

```
(and (?x Student) (?y Faculty) (?z Course)
     (?x ?y advisor) (?x ?z takesCourse) (?y ?z teacherOf)))
```

Obviously, 6!=720 execution plans exist. In order to measure the effectiveness of the heuristic optimizer, we switched off the optimizer and executed and measured the runtimes of all 720 permutation queries. Moreover, we executed the queries in an incomplete nRQL mode such that *no ABox reasoning is required*. This is necessary in order to *only* measure the effectiveness of just this single optimization technique; thus, approximately constant time is needed for each variable binding test and each variable binding generation step. The size of the LUBM ABox is also not important here, but for the sake of completeness of information we state that we have used 6 university departments (we have kept the size small, since the test takes too long otherwise). The measured and (in ascending order) sorted runtimes are shown in Fig. 5.7.1. The fastest permutation query needed only 0.013 seconds, and the slowest 62.723 seconds. This is a factor of 4824.846. If the optimizer is turned on, then it generates a plan which corresponds to the third best permutation query; we measured 0.03 seconds.

However, since query answering is performed in milliseconds here, this could be noise as well and one could claim that the optimizer selected an "optimal" query. Due to the long runtimes required for this test we did not measure variances. The result shows that this optimization is *good enough* and indispensable.

Another very important exploited optimization technique concerns the computation of the role successors resp. predecessors. Efficient computation of these successors is very important, since the optimizer prefers a "navigational" approach to variable binding computation (since successor generators and predecessor generators of binary atoms are preferred over unary generators).

In principle, an ABox satisfiability check is required in order to check whether j is an R successor of i in $\mathcal{A}$, since $A \models (i,j) : R$ iff the ABox $\mathcal{A} \cup \{i : \forall R.M, j : \neg M\}$ is unsatisfiable, for some fresh marker concept M.

However, for simple DLs without number restrictions or features, an important optimization can be turned on, which we found to be indispensable (e.g., not a single LUBM query can be

answered without this optimization):

Lemma 1 (Syntactic Test for Role Filler Entailment) Let $\mathcal{A}$ be an $\mathcal{ALCHI}_{\mathcal{R}^+}$ ABox, an R be a role. Then, $\mathcal{A} \models (i,j) : R$ iff $\mathcal{A} \rightsquigarrow (i,j) : R$, where $\mathcal{A} \rightsquigarrow (i,j) : R$ iff

- $(i,j) : S \in \mathcal{A}$, $S \sqsubseteq R$, or
- $(j,i) : S \in \mathcal{A}$, for $S \sqsubseteq R^{-1}$, or
- $\mathcal{A} \rightsquigarrow (i,k) : R$ and $\mathcal{A} \rightsquigarrow (k,j) : R$, for some $k \in \mathsf{inds}(\mathcal{A})$, if R is a transitive role. ■

Proof 1 (Sketch) The direction $\mathcal{A} \rightsquigarrow (i,j) : R$ implies $\mathcal{A} \models (i,j) : R$ is trivial (omitted).

In order to show that $\mathcal{A} \models (i,j) : R$ implies $\mathcal{A} \rightsquigarrow (i,j) : R$ one needs to take the properties of $\mathcal{ALCHI}_{\mathcal{R}^+}$ into account. Suppose $\mathcal{A} \models (i,j) : R$, but $\mathcal{A} \not\rightsquigarrow (i,j) : R$. There are two cases. *If R is not a transitive role,* then $\mathcal{A} \models (i,j) : R$ can only hold due to assertions $(i,j) : S$, $(j,i) : S^{-1}$ with $S \sqsubseteq R$, since $\mathcal{ALCHI}_{\mathcal{R}^+}$ does neither offer number restrictions nor functional roles. Thus, also $\mathcal{A} \rightsquigarrow (i,j) : R$ must hold. Contradiction. *If R is a transitive role,* then $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$. If $\mathcal{A} \models (i,j) : R$, then for $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ there must be a path of ABox individuals $k_1, \ldots, k_n$ with $k_1 = i$ and $k_n = j$ of maximum length n, such that all k_i on this path are distinct, and $(k_i^{\mathcal{I}}, k_{i+1}^{\mathcal{I}}) \in R^{\mathcal{I}}$ holds. Since the path is maximal, also $\mathcal{A} \rightsquigarrow (k_m, k_{m+1}) : R$ for all $m \in 1 \ldots n$. However, by definition of $\rightsquigarrow$, we also have $\mathcal{A} \rightsquigarrow (k_l, k_m) : R$ for all $l < m$ and $l, m \in 1 \ldots n$, as well as $\mathcal{A} \rightsquigarrow (k_1, k_n) : R$, so $\mathcal{A} \rightsquigarrow (i,j) : R$. Contradiction. □

Please note that this no longer holds as soon as equality statements (or `owl:same-as`), functional roles or number restrictions are added: Consider $\{(i,j) : F_1, i : \exists F_2.C\}$ with $F_1 \sqsubseteq G$, $F_2 \sqsubseteq G$, such that G is a functional role (feature). Obviously, $\mathcal{A} \models (i,j) : F_2$, but $\mathcal{A} \not\rightsquigarrow (i,j) : F_2$. Another illustrative counter example is given by the ABox $\{(i,j_1) : R, (j_1,j_2) : \mathit{same\_as}, (j_2,j_3) : \mathit{same\_as}, (k,j_3) : R, (k,j_4) : R, k :\leq_1 R\}$ (note that the role *same_as* encodes an equality assertion, or `owl:sameAs`). The Lemma fails completely, but still $\mathcal{A} \models (i,j_k) : R$ for $1 \leq k \leq 4$.

However, for those ABoxes on which the Lemma fails, a weaker statement can be made which states that in order for $\mathcal{A} \models (i,j) : R$ to hold, there must at least be some connecting path of role assertions and/or same-as assertions in the ABox between i and j. So, this can be used as a guard for the required ABox satisfiability tests which reduces the number of tests still tremendously.

Let us demonstrate the effectiveness of these two optimizations. Again, we have used **Q9**. This time we have used MiDeLoRa on a LUBM ABox containing a single university department (it contains 1555 individuals). **Q9** returns 13 result tuples; MiDeLoRa needs 1.3 seconds to compute these if all optimizations are turned on (the system is not yet as performant as RacerPro). The initial ABox satisfiability check still takes 12 seconds, but the subsequent ABox satisfiability checks run faster due to optimizations described in [50, 21]. Such a subsequent ABox test currently needs 3.7 seconds. So, for answering **Q9**, not a single ABox consistency test is performed. We have counted the number of calls to the API function roles_related? initiated by the role tester, role successor and role predecessor generator functions; roles_related? was called 4133 times. So, without the described optimization, 4133 ABox satisfiability tests would be required for answering **Q9**, each one taking approx. 3.7 seconds. This means that 4133 * 3.7 = 4.25 hours would be needed for answering **Q9** with MiDeLoRa. Moreover, the number of roles_related? calls is already minimized, since i and j are only checked for relatedness if there is an appropriate path of role assertions connecting i and j. Thus, a naive implementation would perhaps even generate $1555^2 = 2418025$ calls resp. ABox consistency checks, thus, 103.37 days would be needed. These numbers demonstrate the significance of the optimizations.

Chapter 6

The MiDeLoRa Description Logic System Construction Toolkit

Due to a lack of space we can only discuss some very core design principles of MiDeLoRa. Unfortunately, some basic knowledge on DL tableau prover is required in this section [39, 51].

MiDeLoRa is a toolkit for crafting a DL system. Provers are primarily, but not necessarily, tableau provers. However, also dedicated specialized provers such as instance retrieval provers can be implemented with the provided software abstractions. The central abstraction is the notion of the three-dimensional MiDeLoRa space $\mathcal{S}\times\mathcal{L}\times\mathcal{T}$, as already introduced in Section 5.3. Substrate-specific, language-specific, and task-specific provers cover regions in this space and are provided as ternary Clos multi-methods (thus, they are polymorphic / do late binding w.r.t. to their three main arguments: Substrate, Language, Task). MiDeLoRa is implemented fully object-oriented.

Since an ABox is based on the object-oriented graph substrate data model (e.g., nodes, edges, labels are instances of Clos classes), for reasons of orthogonality we have chosen to also use this representation for the tableau representation (the tableau is the working data structure of a tableau prover). Thus, the system uses "boxed" data structures for tableau representation. This is a high burden, since modifying operations on the tableau as performed by the tableau rules during the proof must be performed destructively then. If *backtracking* occurs during the tableau expansion, then some of these destructive changes to the tableau must be undone, and the tableau state be reverted. Since the used data structures are heavyweight, it is impossible to work with "copies" here.

The system thus keeps a so-called *log buffer* of the operations which have been performed on the tableau. If a clash is encountered, this log is used to "roll back" the tableau; again, to do so, destructive operations must be performed (see also the *chain of commands pattern* in [40]). To the best of our knowledge, MiDeLoRa is the first DL system which uses this technique.

This technique has some drawbacks, especially if a lot of backtracking occurs during a tableau proof. However, our performance measurements indicate that the additional overhead is still acceptable. Benchmarks will be published in future work. From our point of view, the advantages of this heavy-weight explicit representation of the complete prover state as data structures outweigh the disadvantages, since it opens up additional potentials. For example, the prover state can be made persistent at any time.[1] Thus, MiDeLoRa is the first system that can in principle be interrupted during a proof and resume a proof later on (in principle, even on a different machine). This would make the system truly transactional.

Tableau provers are very complex software artifacts; especially due to the exploited optimization methods. Maintainability and reusability is thus a prime requirement, and appropriate software abstractions are required in order to hide complexity and achieve maintainability. MiDeLoRa offers *domain specific languages (DSLs)* for prover construction. The ultimate goal of these software abstractions is to provide conciseness and comprehensibility. With MiDeLoRa it is possible to craft tableau provers which are nearly as concise and clear as the mathematical tableau calculi.

[1] A Common Lisp/Clos serializer developed by the authors makes this possible; the same serializer is also used for storing/restoring the KB images in RacerPro.

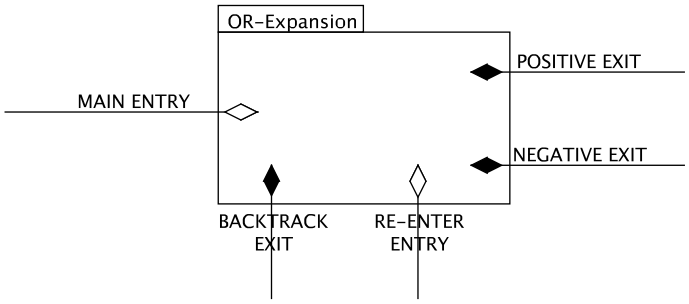

Figure 6.1: 5-Port-Model of a MiDeLoRa Tableau Rule

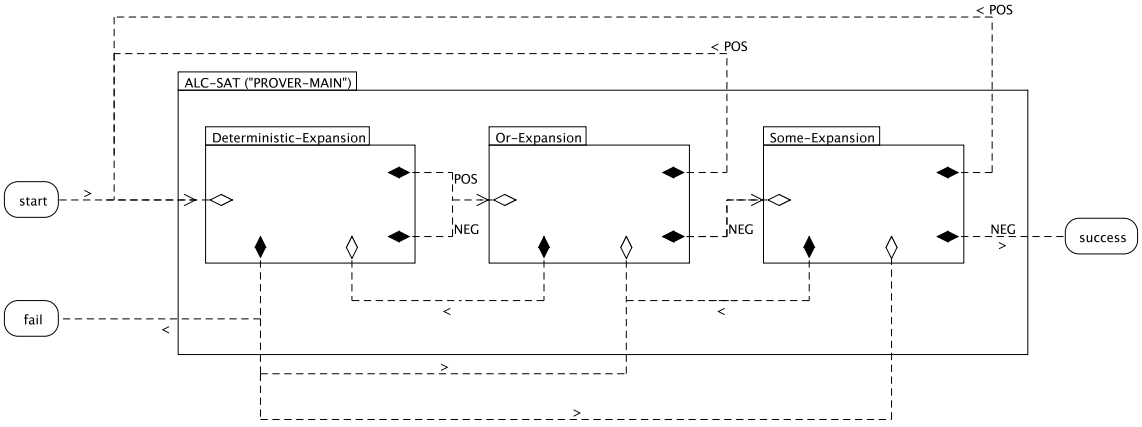

Figure 6.2: MiDeLoRa Model of an $\mathcal{ALC}$ Prover

Thus, it provides abstractions for: tableau rules, strategies, tableau calculi (provers). Rules are designed according to a 5-port model which can be depicted graphically, Fig. 6.1 illustrates the 5-port-model of a simple rule. The ports of rules can then be "wired" together resp. connected to create provers, implementing certain strategies. Since the individual rules are highly optimized, the provers created in such way exhibit a good performance (somehow comparable to state of the art reasoners). A simple $\mathcal{ALC}$ prover is shown in Fig. 6.2.

Rules and provers are reusable, extensible and parameterizable software components. Common Lisp is an ideal language for implementing such DSLs, since it offers unique language extensibility facilities by means of macros (e.g., definition macros such as `defrule, defprover` are provided; within these definitions, no Common Lisp is visible).

Bibliography

[1] T. Berners Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American, 5:2001.

[2] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Description logic framework for information integration, in: Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98).

[3] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Knowledge representation approach to information integration, in: Proc. of AAAI Workshop on AI and Information Integration, 1998.

[4] A. Borgida, M. Lenzerini, R. Rosati, Description logics for databases, in: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[5] H. Wache, V. ogele, T. Visser, U. Stuckenschmidt, H. Schuster, G. Neumann, H. ubner, Ontology-based integration of information - a survey of existing approaches (2001).

[6] M. Wessel, Some Practical Issues in Building a Hybrid Deductive Geographic Information System with a DL Component, in: F. Bry, C. Lutz, U. Sattler, M. Schoop (Eds.), Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB2003).

[7] M. Wessel, R. Möller, A Flexible DL-based Architecture for Deductive Information Systems, in: G. Sutcliffe, R. Schmidt, S. Schulz (Eds.), Proc. IJCAR-06 Workshop on Empirically Successful Computerized Reasoning (ESCoR), 2006.

[8] I. Horrocks, D. L. McGuinness, C. A. Welty, Digital libraries and web-based information systems (2003), in F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[9] V. Haarslev, M. Wessel, GenEd – An Editor with Generic Semantics for Formal Reasoning about Visual Notations, in: 1996 IEEE Symposium on Visual Languages, 1996.

[10] R. Möller, K. Hidde, R. Joswig, T. Mantay, B. Neumann, Bericht über das projekt band: Benutzeradaptiver Netzinformationsdienst, Tech. Rep. LKI-Memo LKI-M-99/01, Labor für Künstliche Intelligenz, Fachbereich Informatik, Universität Hamburg, 1999.

[11] T. R. Gruber, A Translation Approach to Portable Ontologies., Knowledge Acquisition 5 (2) (1993).

[12] V. Haarslev, R. Möller, RACER System Description, in: Int. Joint Conference on Automated Reasoning, 2001.

[13] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive description logics, in: H. Ganzinger, D. McAllester, A. Voronkov (Eds.), Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99).

[14] V. Haarslev, R. Möller, RACER System Description, in: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001).

[15] D. Calvanese, G. De Giacomo, M. Lenzerini, What can knowledge representation do for semi-structured data?, in: Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98).

[16] F. Manola, E. Miller, Rdf primer, Tech. rep., World Wide Web Consortium (Feb. 2004).

[17] D. Brickley, R. V. Guha, Resource description framework (rdf) schema specification 1.0, Tech. rep., World Wide Web Consortium (Mar. 2000).

[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, DL-Lite: Tractable description logics for ontologies, in: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005).

[19] V. Haarslev, R. Möller, M. Wessel, Description logic inference technology: Lessions learned in the trenches, in: I. Horrocks, U. Sattler, F. Wolter (Eds.), Proc. International Workshop on Description Logics, 2005.

[20] R. Möller, V. Haarslev, M. Wessel, On the scalability of description logic instance retrieval, in: Proc. of the 2006 International Workshop on Description Logics DL'06, 2006.

[21] R. Möller, V. Haarslev, M. Wessel, On the scalability of description logic instance retrieval, in: C. Freksa, M. Kohlhase (Eds.), 29. Deutsche Jahrestagung für Künstliche Intelligenz, 2006.

[22] S. Bechhofer, R. Möller, P. Crowther, The DIG Description Logic Interface, in: Proceedings of the International Workshop on Description Logics (DL-2003).

[23] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, T. Weithöner, DIG 2.0 Towards a Flexible Interface for Description Logic Reasoners, in: B. C. Grau, P. Hitzler, C. Shankey, E. Wallace (Eds.), In Proceedings of the second international workshop OWL: Experiences and Directions, 2006.

[24] V. Haarslev, R. Möller, M. Wessel, Querying the Semantic Web with Racer + nRQL, in: Proc. of the 2004 Description Logic Workshop (DL 2004), 2004.

[25] V. Haarslev, R. Möller, M. Wessel, A High Performance Semantic Web Query Answering Engine, in: Proc. of the 2005 Description Logic Workshop (DL 2005), 2005.

[26] A. Newell, The knowledge level, Artificial Intelligence 18 (1) (1982) 87–127.

[27] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, AL-log: Integrating datalog and description logics, Journal of Intelligent Information Systems 10 (3) (1998).

[28] A. Y. Levy, M.-C. Rousset, CARIN: A representation language combining horn rules and description logics, in: European Conference on Artificial Intelligence, 1996.

[29] I. Horrocks, S. Tessaris, Querying the Semantic Web: a Formal Approach, in: I. Horrocks, J. Hendler (Eds.), Proc. of the 2002 International Semantic Web Conference (ISWC 2002).

[30] B. Glimm, I. Horrocks, Query Answering Systems in the Semantic Web, in: Proc. of the KI-04 Workshop on Applications of Description Logics 2004, ADL '04, 2004.

[31] E. Karabaev, C. Lutz, Mona as a dl reasoner, in: Proceedings of the 2004 International Workshop on Description Logics (DL2004).

[32] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, R. Rosati, QuOnto: QUerying ONTOlogies, in: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), 2005.

[33] V. Haarslev, C. Lutz, R. Möller, A description logic with concrete domains and role-forming predicates, J. of Logic and Computation 9 (3) (1999) 351–384.

[34] C. Lutz, M. Milicic, A Tableau Algorithm for DLs with Concrete Domains and GCIs, in: Proc. of the International Workshop on Description Logics, 2005.

[35] M. Wessel, Qualitative Spatial Reasoning with the $\mathcal{ALCI}_{\mathcal{RCC}}$-family – First Results and Unanswered Questions, Tech. Rep. FBI–HH–M–324/03, University of Hamburg, Computer Science Department, available at `http://kogs-www.informatik.uni-hamburg.de/~mwessel/report7.{ps.gz | pdf}` (May 2003).

[36] C. Lutz, F. Wolter, Modal logics of topological relations, in: Proceedings of Advances in Modal Logics 2004, 2004.

[37] B. McBride, Jena: A Semantic Web Toolkit, IEEE Internet Computing 6 (6) (2002) 55–59.

[38] G. L. Steele, Jr., Common Lisp – The Language, Second Edition, Digital Press, 1990.

[39] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook – Theory, Implemenation and Applications, Cambridge University Press, 2003.

[40] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[41] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, U. Mehlig, T. Neumann, M. Völter, U. Zdun, Software-Architektur – Grundlagen – Konzepte – Praxis, 1st Edition, Elsevier Spektrum Akademischer Verlag, 2005.

[42] K. Clark, Negation as failure logic and databases, in: H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum, 1978, pp. 293–322.

[43] O. Kutz, C. Lutz, F. Wolter, M. Zakharyaschev, E-Connections of Abstract Description Systems, Artificial Intelligence 156 (1) (2004) 1–73.

[44] D. A. Randell, Z. Cui, A. G. Cohn, A Spatial Logic based on Regions and Connections, in: B. Nebel, C. Rich, W. Swartout (Eds.), Principles of Knowledge Representation and Reasoning, 1992.

[45] M. Wessel, On Spatial Reasoning with Description Logics - Position Paper, in: I. Horrocks, S. Tessaris (Eds.), Proceedings of the International Workshop on Description Logics 2002 (DL2002).

[46] RacerPro User's Guide 1.9.0, Tech. rep., Racer Systems GmbH & Co. KG, `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf` (2006).

[47] R. Möller, M. Wessel, Terminological default reasoning about spatial information: A first step, in: Proc. of COSIT'99, International Conference on Spatial Information Theory (COSIT), 1999.

[48] M. Wessel, R. Möller, A High Performance Semantic Web Query Answering Engine, in: Proc. International Workshop on Description Logics, 2005.

[49] Y. Guo, Z. Pan, J. Heflin, An evaluation of knowledge base systems for large owl datasets, in: Proc. of the Third Int. Semantic Web Conf. (ISWC 2004), 2004.

[50] V. Haarslev, R. Möller, Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results, in: Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004.

[51] F. Baader, U. Sattler, Tableau algorithms for description logics, in: R. Dyckhoff (Ed.), Proc. of the 9th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000).

Part IV

Scalable and Efficient Reasoning Infrastructures

Chapter 7

Semantic Middleware

One of the most exciting promises of the Semantic Web is to enable the development of software agents that can autonomously accomplish sophisticated, user-specific tasks. In order to do this, software agents have to execute complex queries that may require machine-based reasoning in order to derive implicit knowledge. The information relevant for query answering in this context is usually located in several ontologies (also known as knowledge bases, KBs) that do not share a common vocabulary and are distributed over the Semantic Web. In the context of the BOEMIE project, agents might ask servers for specific media objects.

From a slightly generalized point of view, in this work we consider software applications which in parallel generate queries w.r.t. many different "external" KBs. We presuppose that each KB is available from one or many so-called KB or ontology server(s). Furthermore, for a particular KB there may exist many possible reasoner systems that can answer queries over it. We concentrate on scenarios where queries are posed to a single KB and leave out other settings which are on the focus of established research fields like ontology integration and data exchange.

In our view, the KB servers we consider are managed by different organizations and, maybe in the near future, each transaction (or query) requires some payment in case of a commercial environment. Therefore, Semantic Web applications used in some company need some gateway inference server that provides local caching (in the intranet) to: (i) reduce external communication and (ii) avoid repetitive external server access operations in case multiple intranet applications pose the same queries.

In order to successfully build these kinds of applications, an appropriate "semantic middleware" is required that is responsible for organizing the access to multiple reasoners working in parallel. If there are many reasoners available for a specific KB, the task of the middleware is to manage request dispatching and load balancing. This is particularly important if we consider scalabilitiy issues for BOEMIE application scenarios such as information systems for media objects. Note that in contrast to database systems, at the current state of the art, a single reasoner does not support transaction processing. As a result, processing multiple queries at a time does not lead to reduced answering times compared to the sequential execution of the queries.

As we will argue below, the design of a middleware architecture concerning optimization techniques must be tailored toward the specific scenario sketched above. The middleware is expected to run at the clients' site (in the intranet), and queries are delegated to remote reasoners. Hence, load balancing must be accompanied by middleware-side caching in order to reduce network latency. In order to appropriately support external services such as payment gateways, the architecture must be based on standard software components such as application servers and must provide for standard interfaces.

The contribution of this chapter is (i) to develop a software architecture for systematically combining multiple Semantic Web reasoning engines and (ii) to investigate associated problems of load balancing and caching strategies (supporting multiple users of the local site). Caching is investigated in the presence of incrementally answered queries. In addition, the effects of concurrent query executions on multiple (external) inference servers and corresponding transmissions of multiple partial result sets for queries are studied.

The rest of this chapter is structured as follows. We first discuss the specific requirements

for the middleware with regard to the issues raised in the Semantic Web context. Afterwards, optimization criteria have to be exploited by the middleware are described. Focusing on specific aspects of the implementation of the architecture we then introduce core algorithms that allow for performance improvements for multiple clients. We then evaluate the proposed architecture using practical experiments with the particular implementation. Next, we propose further substantial enhancements for the middleware architecture. We conclude with the summary of the chapter and discuss ideas for future work.

7.1 Requirements for the Semantic Web Middleware

In the context of Semantic Web applications, the following requirements have to be met. It should be mentioned that some of them are similar in database systems, but with Semantic Web-specific peculiarities.

7.1.1 Iterative Query Answering

Modern reasoners (e.g., RacerPro [4]) support iterative query answering, where clients may specify the maximum number of answers they want to get from the server, and thus servers return partial answer sets. For iterative query answering, the reasoner can be configured to compute the next tuples on demand (lazy mode) or in a concurrent way (proactive mode). Moreover, it can be instructed to return "cheap" (easily inferable from syntactic input) tuples first [5].

By using available configuration options, such as incomplete modes, that are powerful enough for semi-structured ontologies, the reasoner achieves significant performance improvements for handling large ontologies. However, this task still demands reasonable hardware resources and is notably memory-intensive.

In the Semantic Web scenario, the middleware is requested to provide for efficient iterative query answering. Thus the middleware has to exploit the corresponding functionality and configuration options (such as cheap tuples first mode) offered by reasoners.

7.1.2 Handling of Concurrent Client Requests

Although highly-optimized reasoners can achieve significant performance improvements for a single client, this effect decreases in scenarios where multiple clients pose queries concurrently. In fact, a single reasoner instance cannot process several client requests in parallel (although they can have multiple open queries for which further tuples can be retrieved in a subsequent incremental query). Thus, as long as the reasoner is processing a query (which usually includes activities such as parsing the query, reading the corresponding knowledge base, classifying it, as well as finding the requested number of answer tuples and returning them). All other clients have to wait in a queue.

Due to this observation, we claim that one of the main requirements for the Semantic Web middleware is the ability to handle concurrent client requests. For that, the Semantic Web middleware has to act as a proxy that employs sophisticated dispatching algorithms in order to delegate queries to multiple back-end DL reasoners. In this setting, the DL reasoners work dedicatedly in the sense that they exclusively work for the middleware. Hence the middleware can easily monitor the current state of each dedicated reasoner and can decide how to dispatch concurrent client requests optimally.

7.1.3 Scalability and Efficiency

Another key requirement for the Semantic Web middleware is the capability to scale up in order to support scenarios that cause high query traffic. The required scalability can be achieved by increasing the number of reasoners managed by the middleware. In situations where a single middleware may itself become a bottleneck, one possible solution would be to replicate it and to set up a HTTP or hardware dispatcher in front of multiple middleware instances. This way the middleware allows for building modular and hence scalable reasoning infrastructures for Semantic Web applications.

Additionally, the middleware is required to minimize the answering time of each query, which can only be achieved by operating as efficiently as possible. It is obvious that for a single query the middleware can not accelerate the computation of query answers. However if queries are posed concurrently, which regularly occurs in the Semantic Web scenario, the middleware can enable the answering of queries in parallel and hence improve the overall efficiency. Furthermore, it can exploit several optimization criteria to reduce the number of inference service invocations and to find the most appropriate reasoner that can answer a query in the shortest time. We discuss details of optimization criteria in the next section.

7.1.4 Support for Standard Query Languages

The Semantic Web middleware has to implement widely used and accepted standard components and protocols such as interfaces, query languages and communication protocols in order to enable the interoperability of a wide range of applications. Consequently, we argue that service-oriented architectures and web services are first class choices for Semantic Web middleware. Considering a query language and communication protocol, we decided on the OWL Query Language (OWL-QL), which is a candidate standard for instance retrieval [3]. OWL-QL defines a protocol for query-answering dialogs among agents using knowledge represented in the Web Ontology Language (OWL [9]). It facilitates the communication between querying agents (clients) and answering agents (servers). Due to the fact that a vast amount of information will be available in many different formats on the Semantic Web, OWL-QL also supports features known in traditional database query languages such as SQL. For instance, OWL-QL allows clients to request partial sets of answers.

It should be mentioned that the OWL-QL standard does not specify anything about server-side implementation details such as caching, number of reasoners used as back-end etc. However, the OWL-QL protocol for query-answering dialogs and the heterogeneous nature of the Semantic Web itself makes it obvious that the middleware which claims to serve as an OWL-QL server has to manage multiple reasoners in the background.

With regard to the near future of modern DL systems, the existing standard interface for accessing DL reasoners (DIG) will become more important not only as a communication protocol but also as a query language. The upcoming version, namely DIG 2.0 [12], offers many essential features such as iterative query answering and query management. Thus it is realistic to expect that DIG 2.0 will replace OWL-QL as a standard query language.

7.1.5 Integration with Other Software Components

Another important requirement for the middleware is to provide for easy integration into existing infrastructures such as firewalls, application servers or billing systems. This can be achieved if the middleware offers standard interfaces and benefits from a multi-layered architecture. Consequently, Semantic Web applications exploiting the proposed middleware can benefit from further services such as security, trust, accounting etc.

7.2 Optimization Criteria

The primary goal of the Semantic Web middleware is to minimize the query answering time for each client in a scenario where multiple clients pose queries concurrently. To do this, it exploits several optimization criteria when searching for the best decision on how to answer a query as quickly as possible. Next we discuss these optimization criteria in detail.

7.2.1 Query Classification

We start with query classification that is crucial for choosing the right optimization strategy according to criteria, which are described next. Due to the incremental nature of queries we suppose, queries are categorized into the following four classes:

- **First query to an unknown KB**: The instances of this query class reference a KB, which is unknown to the middleware. An unknown KB is a KB that has not been loaded by any of the reasoners managed by the middleware.

- **New query to a known KB**: New queries are queries that reference a known KB and are posed to the middleware for the first time.
- **Known query to a known KB**: Known queries are queries that also reference a known KB but have been posed to the middleware before.
- **Continuation query**: Continuation queries are queries posed by clients to get more answers to a query they have posed previously. This type of queries only play a role if clients want to get additional chunks of answer tuples.

7.2.2 Cache Usage

In the Semantic Web scenario, reasoning is an expensive task that requires considerable system resources and time. Nowadays, most reasoners already implemented some efficient caching mechanism. However, if a new layer is involved to build up a reasoning infrastructure, namely a middleware that mediates between clients and reasoners, it is much more efficient to cache inferred knowledge in this layer. In this case queries are answered from the cache whenever possible. E.g., if a query is classified as a known query (to a known KB), the cache must contain at least some of the answer tuples to this query. Only if the required number of answers cannot be delivered completely from the cache, a reasoner will be instructed to deliver the missing number of tuples. Hence, caching in the middleware tier will avoid unnecessary communication with reasoners. Furthermore, clients will benefit from knowledge gained through queries posed by other clients. The positive effects of the cache usage will increase the more clients interact with the middleware.

7.2.3 Query Subsumption

DL reasoners like RacerPro can classify queries in a subsumption hierarchy. Given a new query (to a known KB), the computed subsumption hierarchy w.r.t. queries, which have been answered from the same KB previously, can be exploited to reduce the query answering time.

If the new query is subsumed by one of the previous queries, the search space for answers to the new query can be narrowed down from the whole domain to the answer set of the previous query. This means that the reasoning process can benefit from the subsumption hierarchy to reduce the computation effort and hence answer the query faster. To profit from this effect, a reasoner that already processed a query subsuming the current query becomes the preferred candidate to answer the current query.

In the opposite case, namely if the new query subsumes a previous query, answers of the previous query are obviously also answers for the new query. Therefore, the middleware can first deliver answers from the cache of the previous query. In fact, if the new query requires answers incrementally and the number of requested answers doesn't exceed the cache size of the previous query, the middleware can answer the query without any communication with remote reasoners. It should be noticed that the time spent on the computation of the query subsumption hierarchy is ignorable short compared with the time needed for query answering on large KBs.

7.2.4 Load Balancing

In some situations the middleware can uniquely determine the most appropriate reasoner to answer a query by exploiting the criteria discussed so far. This can be the case, for example, if a query is classified as a continuation query (and thus containing a reference to the reasoner that has already returned some answers to it) or if a query is a new query to a known KB that has been loaded only by a a single reasoner.

However, there are several other scenarios where more than one reasoner can answer a query efficiently and the middleware has to make a decision. E.g., this is the case, when a query references a known KB that has been loaded by more than one reasoners or it references an unknown KB and several idle reasoners could first load the KB and then answer the query. In such situations, the middleware should take into account the current load of each reasoner and try to distribute the reasoning tasks as homogeneously as possible. The better the middleware can balance the load among reasoners the better it will be prepared for future queries.

7.2.5 Knowledge-Base Distribution

Considering the load balancing criteria it is desirable for the middleware to have as many as possible candidate reasoners for answering a query. An important precondition for a reasoner to become a candidate to answer the query is that the reasoner already has loaded the KB referenced by the query. If a reasoner does not "know" the KB, it first has to load it which can take quite some time depending on unknown parameters such as size of the KB and network latency. Therefore, the performance of the middleware can be improved substantially if reasoners are prepared for query answering in advance, namely had loaded the KBs, classified the terminologies and constructed the index structures for query answering.

For that, the middleware can distribute known KBs to idle reasoners that have not loaded these KBs yet. Following a naive approach the middleware could try to distribute every known KB to every reasoner. However, a reasoner has only a finite amount of computational resources at its disposal and moreover some KBs may be only rarely requested. Considering these aspects, it would make more sense to let only "relevant" KBs be pre-loaded by the reasoners, where a measure for KB "relevance" has to be defined.

Chapter 8

RacerManager: An OWL-QL Server as Semantic Web Middleware

In order to support the thesis that despite its highly distributed nature, reasoners in the Semantic Web can be appropriately integrated using specific middleware, we developed an open source system called RacerManager[1], which acts as an OWL-QL server. RacerManager (henceforth called an OWL-QL server) passes the queries to remote or local DL-based reasoners that manage the KB referenced in the query and load KBs from ontology servers on demand. Figure 8.1 illustrates this scenario.

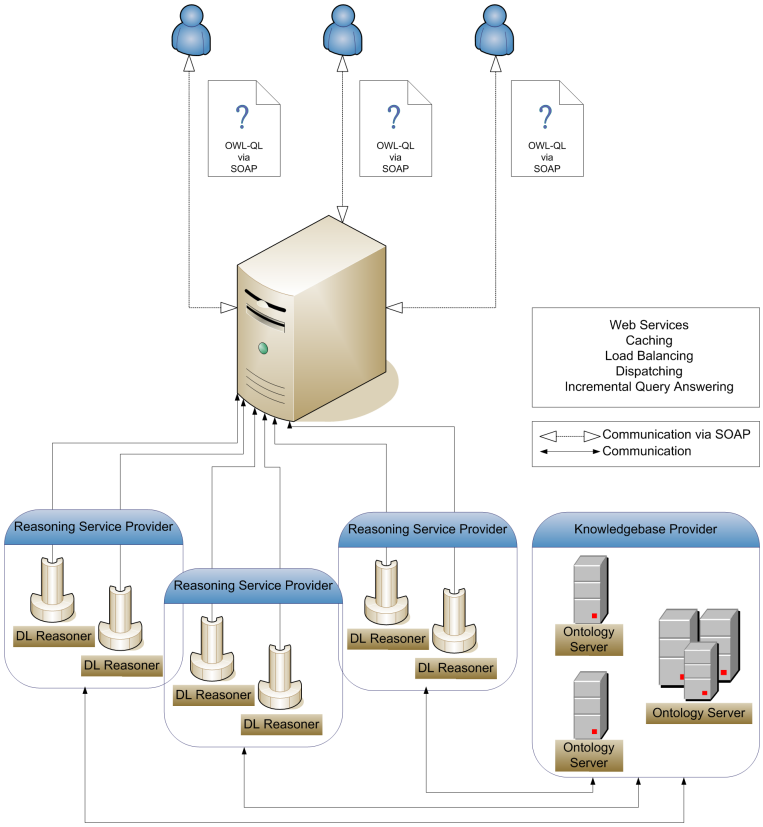

Figure 8.1: OWL-QL Server in the Semantic Web Scenario

[1]RacerManager can be found under: http://www.racerproject.sourceforge.net

Our OWL-QL server is implemented in Java and integrates several widely used components and systems, such as RacerPro as DL reasoner, Tomcat as application server/servlet container [11], Apache Axis Web Services Framework [1], XMLBeans Framework [13] and Jena Semantic Web Framework [7].

We chose a common n-tier architecture as the base layout for the system architecture. This is shown in Figure 8.2. This design provides a clear separation of responsibilities, easy extensibility by modification of single elements and a defined message flow through the system.

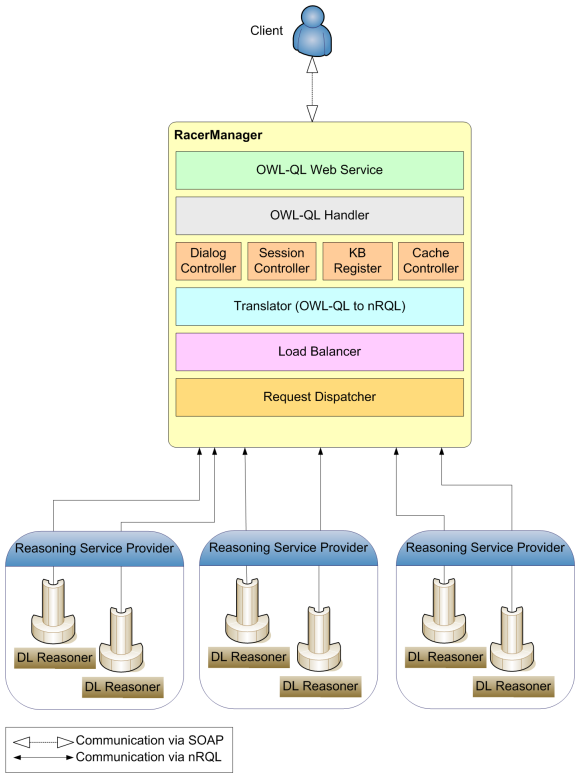

Figure 8.2: RacerManager Architecture

RacerManager initializes and manages an array of reasoners and offers a web service interface for communication with clients. With respect to the states of the managed RacerPro instances and a simple load-balancing strategy (similar to round-robin), RacerManager dispatches the queries to RacerPro instances.

Next, we discuss the central features of the algorithms implemented in RacerManager in detail.

8.1 Caching

Due to the vague nature of the OWL-QL specification, the OWL-QL server might or might not cache query results. However, RacerManager caches each query sent by clients and each answer to that query gained through reasoning. Furthermore, OWL-QL allows clients to specify the maximum number of tuples they are interested in for a certain query. The retrieval of further tuples for the same query is possible in subsequent requests (OWL-QL calls this a dialog). Whenever a client specifies the (maximum) number of tuples for a query, the OWL-QL server returns a unique query identifier, a so-called process handle. Afterwards the client can request more tuples by referring to the received process handle. The handle identifies a query and a client-specific pointer to the element in the result set (seen as a cache array) that previously was the last tuple being returned. In other words, the middleware manages the lifecycle of query dialogs (client sessions). Multiple sessions for the same query term refer to the same cache array. If the same query is posed by another client, the middleware first looks up in the cache of the corresponding query-dialog to

answer the query. Only if the required number of answers cannot be delivered completely from the cache, the corresponding reasoner will be contacted and only the missing tuples will be requested. Details of the cache usage for the retrieval of answer tuples are presented in Algorithm 1.

Algorithm 1 *retrieve_answer_tuples*(*num_req_tuples*, *handle*):

```
result := {}, new_tuples := {}, cached_tuples := {}
current_cache := cache_array(handle) // retrieval ops refer to current_cache
if (num_req_tuples + ind_last_tuple(handle)) > ind_last_tuple_in_cache then
   num_new_tuples := num_req_tuples + ind_last_tuple(handle) − ind_last_tuple_in_cache
   new_tuples := retrieve_new_tuples(reasoner(handle), num_new_tuples, current_cache)
   // We assume that retrieve_new_tuples() adds new tuples to current_cache.
   // retrieve_new_tuples() wait until the reasoner accepts requests.
   cached_tuples :=
     retrieve_cached_tuples(ind_last_tuple_in_cache − ind_last_tuple(handle), current_cache)
   result := new_tuples ∪ cached_tuples
   ind_last_tuple_in_cache := ind_last_tuple_in_cache + num_new_tuples
else
   result :=
     retrieve_cached_tuples(ind_last_tuple_in_cache − (num_req_tuples + ind_last_tuple(handle),current_cache)
end if
ind_last_tuple(handle) := ind_last_tuple(handle) + num_req_tuples
return  result
```

Caching can result in extraordinary resource consumption. Thus, intelligent cache management strategies are required. In our experiments we used a setting, where cache expiration can be set to a particular duration or completely turned off. In the latter case, the cache will never expire. The described caching mechanism can reduce communication overhead with reasoners, especially if they reside on remote servers, and therefore will improve the overall system performance. This holds particularly if one cache array is used for multiple clients posing the same query term. The procedure shown in Algorithm 1 can be extended to the case where the reasoner for a particular query answering dialog is busy (answering other queries, see below). Then it is possible to acquire another reasoner and start with a new empty cache array.

8.2 Request Dispatching and Load Balancing

On the one hand, in the context of business applications using database replication, queries are dispatched to a database instance with respect to the chosen load balancing strategy. On the other hand, instructions that require a change in data must be propagated to all database instances. To enable this, some databases, transaction processing monitors, and application servers offer different mechanisms such as distributed commit protocols.

Regarding Semantic Web applications using OWL-QL for querying, clients can merely query KBs but cannot modify them. Despite that, iterative query answering combined with dispatching and load balancing is more complex than it seems at first sight. We have to consider the fact that in queries, OWL-QL clients can dynamically reference new KBs. Therefore, an OWL-QL server has to track the state of each reasoner it manages. Using this information, the OWL-QL server can decide where to dispatch a query and can balance the load in a better way. The middleware can even allocate additional reasoners and instruct them to load a KB to which an incoming query refers (load on demand).

Whenever a query arrives at the OWL-QL server, the server has to check if it already processed this query and if some of the answers are already in the cache of the corresponding dialog. First, the server checks its internal repository in order to find out if the knowledge base referenced by the query is already loaded by one of the managed reasoners. This possibly means that the server already answered some queries from this knowledge base. In this case it has to search for a dialog associated with the same query. If such a dialog exists, the necessary answers will be taken from the cache. See Algorithm 1 for details of cache usage. It is obvious that there is no point in searching for existing dialogs if none of the reasoners has the required KB loaded, i.e., if a KB is completely new to the middleware. In this case, or if there exists no dialog with the same query (and a known KB), a new dialog will be created (see Algorithm 2).

After a new dialog is created or an existing dialog is assigned, the server has to dispatch the query to an appropriate reasoner. The reasoner that already returned some answers to this query or at least loaded the referenced knowledge base is preferred. If no such reasoner can be found,

Algorithm 2 *retrieve_dialog*(*query*):

```
dialog := null
reasoner := null
if ¬referenced_KB_loaded(query) then
   reasoner := get_next_idle_reasoner()
   reasoner.load_KB(query)
   dialog := create_dialog(query, reasoner)
   return dialog
else
   dialog := search_dialog(query)
   if dialog ≠ null then
      return dialog
   else
      reasoner := get_reasoner_with_referenced_KB(query)
      return create_dialog(query, reasoner)
   end if
end if
```

the server has to delegate the required reasoning task to an idle reasoner with respect to some load balancing strategy (e.g., round robin). After an idle reasoner is found, the reasoner must be instructed to load the referenced knowledge base first, and then to reason over it in order to return the required number of answers.

8.3 Experiments

In order to demonstrate one of the major benefits of the middleware w.r.t. large knowledge bases and multiple clients querying the system concurrently, we carried out some experiments with RacerManager.

In this preliminary experiment, RacerManager is configured to manage two RacerPro instances: one on the local host and another one on a remote machine. Moreover, two clients are involved in the experiment. Each one poses a query regarding a sample university ontology, which is generated using the tools provided by the Lehigh University Benchmark and includes a large data repository with more than 20000 individuals. Note that the sample university ontology has been loaded by both RacerPro instances during startup. The clients interact with RacerManager in the following order: At first Client 1 sends a query that requires computation-intensive inference in order to retrieve more than 5000 answer tuples from the university ontology. Shortly after that, Client 2 sends a query that requires a very short computation time and results in a small amount of answer tuples from the same ontology.

The debugging information shows that RacerManager dispatches the queries to different RacerPro instances and hence enables the concurrent processing of the requests. Consequently, the client with the less demanding query, in this experiment Client 2, receives his results first.

These initial test results reveal that the proposed architecture prevents clients from blocking one another, as it is the case when multiple clients interact with a single reasoner concurrently. Indeed, a detailed analysis which cannot be presented here due to space constraints shows encouraging performance results and indicates that the middleware (i.e., RacerManager) is not a bottleneck.

Currently, there exist some prototypical OWL-QL server implementations, e.g., the Stanford OWL-QL Server [10] and DQL Server developed in Manchester [2]. The Stanford OWL-QL Server uses the first order logic theorem prover JTP [8] and the Inference Web [6] proof exchange system to answer the queries. It supports premises and proofs. The Manchester DQL Server can handle only tree-shaped conjunctive queries. For each new query, it requires all tuples from the reasoner at once and caches them. In contrast to implementations mentioned above, RacerManager focuses on techniques which allow to achieve better scalability, high efficiency and the required quality of service. RacerManager supports a practically very relevant subset of OWL-QL (so-called grounded conjunctive queries) and provides the expressivity of tree-shaped conjunctive queries. Furthermore, it handles queries that include a premise (so-called if-then queries). The server does not support scenarios where multiple knowledge bases are to be used to answer a query or where a knowledge base is not specified by the client.

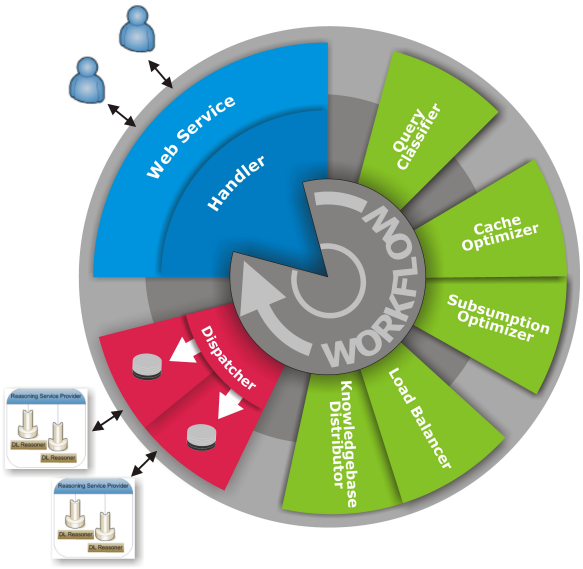

Figure 8.3: Enhanced Architecture for Semantic Web Middleware

8.4 Further Enhancements

Based on the requirements and optimization criteria identified for the Semantic Web middleware and the insights gained through the implementation of an OWL-QL server we propose further enhancements for the architecture. Figure 8.3 depicts an overview of this architecture.

In this section we not only discuss enhancements for existing components of the implementation but also introduce new modules that provide for additional functionality.

8.4.1 The Classification Module

The classification module is a new module that has the task of analyzing incoming queries in order to categorize them into one of the following four disjoint classes: (i) first query to an unknown KB; (ii) new query to a known KB; (iii) known query to a known KB; (iv) continuation query.

The classification of incoming queries is valuable, because each query class uniquely determines the optimization criteria applicable to its instances. Consequently, each query can be assigned with a workflow w.r.t. the query class it belongs to. The procedure followed by the classification module is shown in Algorithm 3.

Algorithm 3 $classify\_query(query)$:

```
workflow := null
dialog := null
if continuation_query_p(query) then
    workflow := create_continuationQuery_workflow(query)
else if ¬referenced_KB_loaded(query) then
    workflow := create_firstQuery_to_unknownKB_workflow(query)
else
    dialog := search_dialog(query)
    if dialog ≠ null then
        workflow := create_knownQuery_to_knownKB_workflow(dialog)
    else
        dialog := create_dialog(query)
        workflow := create_newQuery_to_knownKB_workflow(dialog)
    end if
end if
return workflow
```

8.4.2 Subsumption Optimizer

Subsumption optimizer is another new module that enables the enhanced architecture to benefit from the query subsumption hierarchy. It contains a reasoner that is solely responsible for computing query hierarchies for each KB. This reasoner loads every KB that is known to the system. In particular, it stores the rather small-sized terminological part of every KB and omits the assertional part. It should be noticed that as part of the subsumption optimizer this reasoner runs on the same physical machine and requires few computational resources. As a result, the time and the resources spent for computing the query subsumption hierarchy is ignorable compared to the time spent for the communication overhead with remote reasoners and for query answering on KBs with a large assertional part, which is the usual case in the Semantic Web scenario.

The subsumption optimizer module can reduce answering time of any query in case this query is in a subsumption relation with any former query. In case the new query is subsumed by a previous query answered by some reasoner, the subsumption optimizer sends a message to the dispatching module that this reasoner can answer the new query faster than others. If the new query is subsumed by more than one former queries, the most specific subsumer is used to identify the most appropriate reasoner with the smallest search space for answering the new query.

In case the new query subsumes a former query the subsumption optimizer informs the cache optimizer to deliver the results from the corresponding cache. Details of this new kind of cache usage will be discussed in the following. If the new query subsumes more than one former queries, the one with the largest cache size is preferred.

8.4.3 Cache Optimizer

The existing cache optimizer module can be extended to provide for a new kind of cache usage. The cache optimizer gets a message from the subsumption optimizer in case it has been detected that the new query subsumes a former query. In other words, the new query is more general than a former query and hence, the answers of the previous query are also answers of the new query.

If the number of answers in the cache of the previous query is greater than the number of answers requested for the new query, the new query is answered completely from the cache. Otherwise the missing number of answers is requested for the previous query first. These answers complete the cache of the previous query. If after that answers of the previous query are still not enough for the new query, the new query will be send to a reasoner to get more tuples.

8.4.4 Load Balancer

In the current middleware implementation, the load balancer module makes use of the round robin strategy. In the future, it can be enhanced to take into account the current load of each reasoner in order to distribute the inference tasks as homogeneously as possible. For each reasoner, all requests are collected in a reasoner-specific queue. For computing the current load of a reasoner, the load balancer monitors the state of the corresponding queue and uses several statistics to evaluate and compare it with others (details of this procedure are omitted here due to space restrictions). Using this information, the reasoner with the least load is chosen for the incoming task.

8.4.5 Knowledge-Base Distributor

A new module, the knowledgebase distributor, monitors the relevance of each known KB and instructs idle reasoners to load relevant KBs such that the middleware is optimally prepared for feature queries. The relevance of a KB at some time can be computed by the knowledgebase distributor by taking into account the number of reasoners that already loaded this KB and the amount of queries posed to it so far. By doing this, the module ensures that in the future more reasoners will be available for incoming queries. Therefore the middleware can get closer to its primary goal of minimizing the answering time for each query.

8.4.6 Dispatcher

The existing dispatching module forwards query answering requests to the reasoners immediately. In the enhanced architecture, this module is responsible for managing the above mentioned reasoner-specific queues and distributing requests to the queues. The queues follow a first-come, first served (FCFS) behavior and are unbounded in size. Other modules such as the load balancer have just read-only access to these queues.

Bibliography

[1] Apache Axis Web Services Framework. http://ws.apache.org/axis/.

[2] Manchester DQL server. http://www.cs.man.ac.uk/~glimmbx/.

[3] R. Fikes, P. Hayes, and I. Horrocks. OWL-QL - A Language for Deductive Query Answering on the Semantic Web. Technical Report KSL-03-14, Knowledge Systems Lab, Stanford University, CA, USA, 2003.

[4] 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.

[5] Haarslev Volker, Moeller Ralf, Wessel Michael. The New Racer Query Language-nRQL. 2005.

[6] Inference Web. http://iw.stanford.edu/.

[7] Jena Semantic Web Framework. http://sourceforge.net/projects/jena/.

[8] JTP. http://ksl.stanford.edu/software/JTP/.

[9] Deborah McGuinness and Frank van Harmelen. OWL Web Ontology Language - Overview. W3C Recommendation, http://w3.org/TR/owl-features/REC-owl-features-20040210, 2004.

[10] Stanford OWL-QL server. http://onto.stanford.edu:8080/owql/FrontEnd.

[11] Jakarta Tomcat. http://jakarta.apache.org/tomcat/.

[12] Anni-Yasmin Turhan, Sean Bechhofer, Alissa Kaplunova, Thorsten Liebig, Marko Luther, Ralf Moeller, Olaf Noppens, Peter Patel-Schneider, Boontawee Suntisrivaraporn, and Timo Weithoener. DIG 2.0 – Towards a Flexible Interface for Description Logic Reasoners. In B. Coence Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *OWL: Experiences and Directions 2006*, 2006.

[13] XMLBeans. http://xmlbeans.apache.org/.