Flexible Software Architectures for Ontology-Based Information Systems

Michael Wessel *, Ralf Möller

Hamburg University of Technology (TUHH), Hamburg-Harburg, Germany

Abstract

Although nowadays powerful Semantic Web toolkits exist, these frameworks are still hard to apply for designing applications, since they often focus on fixed representation structures and languages. Prominent examples for applications using Semantic Web representation languages are ontology-based information systems. In search of a more flexible software technology for implementing systems of this kind, we have developed a framework-based approach which is influenced by Description Logics but also supports the integration of reasoning facilities for other formalisms. We claim – and support that claim using case studies – that our framework can cover *regions* in the system design space instead of just isolated points. The main insights gained with this framework are presented in the context of ontology-based query answering as part of a geographical information system.

Key words: Ontology-Based Information Systems, Description Logics, Ontology Query Languages, Semantic Web Software Architectures

1 Introduction

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 (cf. the vision of the Semantic Web [1]), but also play a key role in supporting data and information quality checks, IS interoperability, and information integration [2–4]. Ontologies provide the means for solving problems raised by semantic heterogeneity

* Corresponding author.
Email addresses: mi.wessel@tuhh.de (Michael Wessel), r.f.moeller@tuhh.de (Ralf Möller).

in ISs based on different conceptual or logical data models, because ontologies inherently work on a semantic rather than on a syntactic level and thus support a seamless incorporation of conceptual domain constraints into the machinery of an information system [5].

In this paper we present a formal and implemented generic framework for building ontology-based information systems (OBISs). As such, our framework must offer the means for (i) the extensional layer, (ii) the intensional layer, and (iii) the query component. Being ontology-based, our framework is strongly influenced by Description Logics (DLs) and offers novel solutions for certain problems we have encountered during our endeavor of implementing OBISs with a standard DL system. We make these problems transparent by means of a case study: design and implementation of an ontology-based geographic information system (GIS). Based on our framework we present empirically successful solutions for problems in this specific IS domain. The focus in the case study is on ontology-based query answering. The DLMAPS system supports ontology-based spatio-thematic query answering for city maps [6,7].

Mostly for performance reasons, retrieval systems nowadays still use rather simple thesaurus-based retrieval models (possibly based on statistical information) [8]. From a logical point of view, a thesaurus-based system uses a rather inexpressive representation formalism. Recent developments in Description Logic inference technology have shown that expressive formalisms can indeed be used for building practical systems in general, and practical information systems in particular. For instance, information retrieval systems based on Description Logics are described in [9,10]. In this article, we broaden this view, and describe a framework for building OBISs.

Let us provide some background first. An ontology provides the vocabulary of a conceptualization in a machine-processable format such that the inherent domain constraints and their interrelationships are properly represented. [1] According to Gruber [11], an ontology *is an explicit formalization of such a conceptualization; formal* means that a machine-processable language with formal semantics is used so that the "meaning" is available to the machine. The term *semantic information processing* describes this situation quite accurately, although one could argue that information is "semantic" per se. Ontology-based query answering then means that the vocabulary defined in the ontology can be used in queries to retrieve the desired information from the extensional IS component(s).

DLs are nowadays an accepted standard for *decidable* knowledge representation. It can also be claimed that DLs provide the theoretical foundation for

[1] Unlike in the relational model, where there is usually an information loss when 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.

(formal) ontologies, as well as for the Semantic Web (e.g., the semantics of the Web Ontology Language, OWL, is based on DLs). Usually, in a DL one distinguishes between *ABox* and *TBox*. The ABox is the extensional component, representing information about particular objects of the domain 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* in terms of concepts and roles (unary and binary predicates). From a first-order logic perspective, it contains closed universal first order sentences (axioms). Together, ABox and TBox are called a *knowledge base (KB)*. Inference problems for concepts, roles, and knowledge bases are defined as usual in logic-based formalisms (for details see [12]). The notion of an ontology in the DL sense is a synonym for knowledge base.

For building applications based on Description Logics, an inference engine is needed. Our investigations in this article on building practical information systems are based on the RacerPro engine [13]. RacerPro implements the very expressive DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, also known as $\mathcal{SHIQ}(D^-)$ [14,15], and offers multiple TBoxes, ABoxes as well as expressive concrete domains (of which the OWL "datatypes" are only a subset). Even though it is not clear under which circumstances a reasoning system can be called *empirically successful*, we claim the RacerPro is such a system given the evidence that it has many academic as well as commercial users.

Up to now the number of implemented OBISs is rather small, however. Consequently, experience with the scalability of the DL approach is limited. This is not surprising, since DL systems are a rather new technology compared to databases and, as we will see in the following, some problems remain to be solved in today's DL technology.

1.1 Problem Identification and Motivation for the Approach

We have identified 7 main problems **P1** – **P7** which contribute to the difficulties we encountered regarding the use of DL systems for building OBISs. In this section we identify and describe these problems; later on in Section 2 we describe the pragmatic solutions offered by our framework. The problems **P1, P2** are DL-specific, whereas **P3** – **P6** are specific to the APIs of contemporary DL systems. **P7** concerns the software architecture of DL systems.

We believe that DLs have their deficiencies regarding expressivity and are not a panacea for arbitrary information modeling and representation (of course, this holds for all formalisms). DLs are very well suited for the representation of semi-structured (or even incomplete/uncertain) information [16], but things

become more complicated if special "non-abstract" domains such as space are considered (→ **P1:** DL applicability problem). We say "non-abstract" since space has a rich inner natural structure which is not "man-made", but is given by the laws of physics. Here, either non-standard DLs or non-trivial logical encodings are needed. For these non-standard DLs, no working systems exist, and in our experience, complex logical encodings are very likely to decrease the performance of the reasoning component.

Due to the well-known *expressivity vs. complexity tradeoff,* sound and complete reasoning with expressive DLs is not a trivial task. *Data scalability* is not always easy to achieve for expressive DLs. There exist inexpressive ontology languages such as, for instance, RDF(S) [17–19] which scale well regarding data complexity. However, these approaches fail to scale regarding expressivity and problems then have to be solved outside the information system by resorting to programming. We believe that a generic framework for building OBISs should be parameterizable in both dimensions: data scalability and expressivity scalability (→ **P2**: data and expressivity scalability problem) [20,21]. If high expressivity is required, it should be supported. However, if only low expressivity is needed, then the user should not have to pay the higher price if a reasoner was used which implements a much more expressive logic (see, e.g., the case study in [22]). This implies that a reasoner should be selected that implements *just the required logic*, so that the lower complexity bound is sufficient, and the upper bound is tight.

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 of building a DL system, existing DL systems must be reused and exploited as *componentware* if possible.

Even though standards such as DIG exist [23], it can be observed that for building practical OBIS some API functionality is still missing, only part of which is currently about to be standardized in $\mathrm{DIG}_{2.0}$ [24]. Compared with the APIs found in *relational database management systems (RDMSs)*, one can observe that functionality regarding the management of the physical schema or storage layer of a DL system is missing (→ **P4:** missing storage-layer-functionality problem).

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* is not yet achieved in standard DL systems [6,24] (→ **P5:** extensibility and openness problem). Even though there is an extension proposal for $\mathrm{DIG}_{2.0}$, which we believe is a very promising idea, $\mathrm{DIG}_{2.0}$ still does not support functional-

ity or API functions to be added to a DL system by users (i.e., application builders). It is clear that this problem can only be addressed by some kind of programming facility or plug-in mechanism.

Only recently, expressive *query languages (QLs)* have been investigated and incorporated into DL systems (→ **P6:** missing QL problem) [25–27,7,28]. However, these are indispensable for OBIS.

The last problem (**P7**) is closely related to **P2** and concerns the *software architecture* of a reasoner. Although reasoners implementing highly expressive logics are also capable to processing KBs utilizing only a (less expressive) sublogic, one can sometimes observe that specialized reasoners crafted to support smaller logics perform better than reasoners supporting more expressive logics. From the perspective of the more expressive reasoner, the more efficient (and more specialized) inference algorithm implemented by the less expressive reasoner can be seen as an optimization technique. In principle, the performance of the more expressive reasoner can become comparable once a specialized optimization is built in. With more and more dedicated optimizations, whose applicability must be automatically detected, however, the maintenance of the DL system software becomes a serious problem. We believe that it is important to have *appropriate software abstractions* which help to maintain the software and manage the complexity introduced by language-specific optimization techniques.

Specialized reasoning algorithms are not only needed in order to realize special optimizations, but also to implement certain inference tasks. *From a theoretical perspective,* most expressive DL systems "only" have to implement a reasoner to decide one core inference problem, e.g., an ABox satisfiability checker since the other inference problems are *reducible* to the core problem. However, *from a computational perspective*, this seems inadequate because highly dedicated algorithms for special inference problems have to be used to ensure scalability, e.g., for the *instance retrieval problem* [29]. These algorithms are sometimes even more complex than tableau calculi [30], and thus *deserve a clean separation* from other parts of the system code in order to achieve maintainability. Again, appropriate domain-specific software abstractions are needed. We thus call **P7** the "software-abstraction problem".

1.2 Layered vs. Integrated Approaches for OBIS

Why not simply use an RDMS for the storage layer of an OBIS? This would result in a classical *layered architecture* for an OBIS. From the point of view of the RDMS, the inference algorithms then have to reside in the *application*

layer. [2] Since *ontology-based query answering* requires inference, the assertions in the database are used as input assertions for the inference algorithm. Unfortunately, the question *which assertions to retrieve from the database* can only be resolved at runtime by the inference algorithm itself. But if there is no way to tell in advance which assertions will contribute to the final answer of the query and which will not, then database indices are of no great help in order to reduce the set of candidate assertions to consider. Thus, for expressive ontology languages, a layered architecture results in a lot of communication overhead, and the retrieved candidate results from the database must be combined and reasoned about to get the final query answer. Obviously, it would be better if this computation and integration of required sub-results could be done in the RDMS itself by means of a single query. This is possible as the authors of the QuOnto system have shown, but "only" for rather inexpressive DLs. In the case of QuOnto, ontology-based query answering can be performed by the RDMS query answering engine on its own, since the inexpressivity of the underlying DL makes it possible to expand the original query in such a way that it takes the ontology into account [31]. Thus, no real ABox retrieval algorithms are needed. At the time of this writing, it is not clear whether the rewritten queries are always easy for database engines to deal with. The drawback of this layered approach for OBIS is that it is not obvious how to account for expressivity scalability.

We therefore pursue a *truly-integrated approach* for OBIS, although the implementation burden is very high. This means that the storage layer includes the required inference algorithms as well as the query evaluation engine, in one single component, so that data and index structures can be shared. However, our framework also provides support for the layered approach.

1.3 Contributions and Structure of this Article

The main contribution of this article is the description of the framework and the abstractions it provides. We believe that it is important that these abstractions are understood as abstractions on the *knowledge level* as well as on the *symbol level* (as introduced into AI by Newell [32]). The framework is designed to tackle the identified problems **P1**–**P7** (see Section 2), which provide the motivation for the whole approach. Our framework contains abstractions (and working implementations) to realize (i) the extensional component, (ii) the intensional component, and (iii) the query language component of an OBIS.

It will become clear that for all three areas, highly flexible solutions are needed: for the extensional component, the so-called *substrate data model* is intro-

[2] We think it is unrealistic to assume that a system as complex as a tableaux reasoner can be realized as a stored procedure within a RDMS.

duced, for the intensional level, the MiDeLoRa toolkit for crafting DL systems is presented, and for (iii) the *substrate query language (*SuQL*) framework* is designed. Due to a lack of space, we focus on (i) and (iii). The flexibility of the provided abstractions, and thus of the framework, is empirically demonstrated by means of investigations with specific instantiations of the framework. These instantiations thus support our claim that the framework is generic and can cover regions in the OBIS design space.

The most important instantiation discussed in this paper is the DLMAPS system, which implements ontology-based spatio-thematic query answering in the domain of digital city maps. In this IS domain of digital city maps, we must (a) pragmatically solve the map representation problem, especially regarding the *spatial* and the *thematic aspects* of map objects (these notions are defined in Section 3.1), and (b) provide an expressive *spatio-thematic query language*. This QL must be able to address spatial as well as the thematic aspects of map objects. Due to the inherent complexity of the field, we believe that DL system application studies are valuable per se. In the DLMAPS domain, the situation is even more complicated because of the applicability problem for DLs, which mainly concerns the representation of spatial aspects of maps (which we will call the *spatial representation problem* in the following). Moreover, we present some important optimization techniques which are critical for ontology query answering engines.

This paper is structured as follows. We first describe the overall framework and explain how the identified problems **P1**–**P7** are addressed. In the next section we present the DLMAPS case study. 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 four different representation options for the extensional and query component of the DLMAPS system in our framework. All these have pros and cons, demonstrate the flexibility of the framework and thus support our claim that the framework is generic. Next we describe the substrate QL framework SuQL, which plays a crucial role in this work. The nRQL ABox QL [33,34] is discussed as a concrete instantiation of SuQL. Especially for the DLMAPS system, we show how nRQL can be extended by spatial atoms in order to become a spatio-thematic QL. We then describe indispensable optimizations in the SuQL query answering engine and discuss their effectiveness.

2 An Architectural Framework for Building OBISs

In this section we first describe the framework from the knowledge level perspective [32]. From a logical point of view, a so-called *substrate data model* is

introduced, and the main principles of the associated query language SuQL are presented. We also briefly remark on implementation aspects, the symbol level perspective. We believe that both perspectives on a reasoning system are of equal importance in order to guarantee empirical success. A "good design" should encompass both perspectives in order to avoid performance bottlenecks and impedance mismatches. After having presented the framework, we discuss how the problems **P1** − **P7** are tackled. Please recall that **P1** − **P7** provide the motivation for the whole approach; more precisely, **P1, P2** address the knowledge level, whereas the problems **P3** − **P7** address the symbol level.

We do *not* claim that the substrate model is interesting from a theoretical perspective. Its generic character is of course also its weakness. Thus, it must be *specifically instantiated*. An *instantiation of the model* results in a specific substrate type, e.g. a substrate type *ABox.* The formalization presented here is only as detailed and formally elaborated as is beneficial and required for the description of the semantics of the services, especially of the query answering service. We claim that the presented formalization is sufficient for our purpose.

From the knowledge level perspective, the data model is partially inspired by the work on $\mathcal{E}$-Connections [35], tableaux data structures [14], 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 labeled graphs, defined by means of first order logic, and similar knowledge models have been used in AI since the 1960s [12, Chapter 4] (although the substrate model is primarily an *extensional* knowledge model). SuQL is inspired by [25].

From the symbol level perspective, our approach is related to JENA [36], but we have a somewhat broader scope, and the underlying knowledge (data) models are more general than RDF(S), as will become clear in the following.

2.1 The Knowledge Level Perspective

Formally, we base our framework on a graph-based data model which provides the required flexibility and extensibility for the extensional component, the so-called *substrate data model*. A generic substrate query language called SuQL for this data model provides the required flexibility and extensibility on the QL side. The definition of SuQL is only prepared in this section and continued and elaborated on in Section 5.

The substrate model serves both as a mediator and as an abstraction layer ("semantic middleware"). It enables us to specify and build extensional representation layers for spatial and hybrid representations (see Section 3), and is sufficiently general to also encompass ABoxes and RDF(S) graphs. A substrate is thus 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* $L_V : V \to \mathcal{L}_\mathcal{V}$ *maps nodes to descriptions in an appropriate node description language* $\mathcal{L}_\mathcal{V}$*, and likewise for* $L_E : E \to \mathcal{L}_\mathcal{E}$*, where* $\mathcal{L}_\mathcal{E}$ *is an edge description language.*

If $(i, j) \in E$*, then* j *is called a* successor *of* i*, and* i *is called a* predecessor *of* j*. In case* $R \in L_E((i, j))$*, we can (more specifically) talk of an* R*-successor resp. -predecessor.*

The languages $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are not fixed and can be seen as *subsets of first-order predicate logic, FOPL, (denoted in variable-free syntax)*, e.g., some modal logic, description logic, or 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, $L_V(i)$ and $L_E((i, j))$ correspond to $\{\Phi(C)_{x \leftarrow i}, \ldots, R_1(i, j), \ldots, R_n(i, j)\}$, where $\Phi(C)$ returns the *FOPL standard translation* [12, pp. 50] of the concept C, which is a first order formula with one free variable, x, e.g. $\Phi(\exists R.C) = \exists y R(x, y) \wedge C(y)$.

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 simply define a *substrate with a background theory* (having an additional set of closed FOPL axioms). These additional FOPL sentences are obtained by applying the standard translation to the TBox axioms. This should be clear. We will give a formal example for such a substrate with background theory when we discuss the RCC substrate in Section 4.4.

To get spatial representations, we state that a substrate can also encode geometric / spatial structures using FOPL means. For the DLMAPS system, we assume that the nodes are instances of spatial datatypes (e.g., polygons). Such a geometric substrate is called an *SBox (Space Box).* The geometry of such spatial nodes can be described using an appropriate *(FOPL-based) geometry description language*. However, we do not present these details here.

Unlike for an ABox, it is reasonable to assume for an SBox that its *logical theory is complete,* as there is neither underspecified nor indeterminate information in an SBox. It simply represents "spatial data". Viewed as set of FOPL ground atoms, the SBox is basically isomorphic to its (unique minimal) Herbrand model. On the declarative knowledge level we can simply assume that the well-known *Clark completion axioms* are present [37], and that their impact will be "intrinsically" encoded into the inference procedures defined for an SBox.

Since we simply rely on standard FOPL semantics, everything is well defined. We just inherit the standard FOPL notions of satisfiability, entailment ("$\models$"), etc. The entailment relationship is needed for the definition of the SuQL.

The SuQL framework allows for the definition of specialized substrate QLs, tailored for special substrate classes (e.g., ABoxes, SBoxes). The SuQL framework is based on the general notion of *(ground) query atom entailment.* All that matters here is that a *notion of logical entailment* 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 or two free FOPL variables (we use x and y in the remaining paper for these); the atoms are thus called unary (resp. binary) query atoms. Thus, $S \models P_{x \leftarrow i}$ must be decidable for the unary atom P and the node $i \in V$, and $S \models Q_{x \leftarrow i, y \leftarrow j}$ must be decidable for the binary atom Q and the nodes $i, j \in V$.

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, for example, RCC predicates (see Section 4).

2.2 The Symbol Level Perspective

A substrate is an instance of a Clos (Common Lisp Object System) class [38] – a *substrate class* thus provides the implementation of a *substrate type (or kind).* On the one hand, a substrate is thus a representation on the knowledge level, but on the other hand also – and much more importantly in this work – a structure on the symbol (or implementation) level.

We have already used the phrase *instantiation of the substrate data model* informally. More specifically, from now on this means that a new substrate class is defined (tailored for certain representation tasks) by means of subclassing. In the same sense we are using the phrase SuQL *instantiation* to refer to a specialized substrate QL, e.g., one that offers substrate-specific, tailored query atoms. Last but not least, an *instantiation of the framework* encompasses all kinds of instantiations; for example, the DLMAPS system is an instantiation

which contains specific substrate types and specialized SuQL instantiations.

Since Clos offers multiple inheritance (i.e., allows a class to have multiple parent classes), it becomes possible to define *combinations of substrates.* For example, one can define a substrate class *spatial ABox* having the substrate classes *ABox* and *SBox* as parents. As a result, instances in such a spatial ABox are, on the one hand, *ABox individuals*, and instances of spatial datatypes on the other hand [6]. This can eliminate the need for a *hybrid representation* in favor of an *integrated* representation. However, the substrate data model also supports hybrid representations (see Sections 4.2 and 4.4).

Another important idea is that the nodes and edges in a substrate can be "virtual", i.e., the substrate is simply used as a mediation layer or "facade" that provides a graph perspective on a different representation, e.g. a RacerPro ABox. In this case, the API functions of the substrate just pass through to the API functions of RacerPro. Thus, a substrate class *may or may not* correspond to a physical store.

Not only substrates, but also SuQL query atoms are instances of Clos classes. This enables the definition of the $\models$ relation as a (binary) Clos *multi-method* `substrate-entails-atom-p`. A multi-method is polymorphic (does late binding) according to the types of all its arguments [38], unlike languages like Java, where only the type of the first argument is used for dispatching. Thus, depending on the class of substrate and atom, different inference algorithms will be called for (e.g., a DL system API function in case an ABox is queried, and a geometric algorithm performing some kind of *spatial model checking* if an SBox is queried). Furthermore, *intrinsically encoded axioms* can be taken into account in the implementation of a `substrate-entails-atom-p` method, 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. [3]

The generic SuQL *query answering engine* (see Section 6) immediately supports the evaluation of specialized atoms once a `substrate-entails-atom-p` method is applicable, since there are generic enumerator and tester methods defined. However, these will not exhibit good performance, since they only implement *linear retrieval algorithms* (instances are retrieved using "enumerate and test"). However, good performance can be achieved if *dedicated generators and tester* are defined for specialized atoms. These methods will also exploit indices and caches, and so the performance can be very good as we have demonstrated with the nRQL instantiation. Also the *cost-based* SuQL

[3] However, this is not meant to reopen the "declarative vs. procedural" debate; instead, our framework shows that both approaches can and have to live together well, given that appropriate abstractions are provided which are "on the right level" for both perspectives.

query optimizer (see Section 6.1) is easily configurable (some methods must be overridden).

In order to decide entailment (as needed for query answering), inference algorithms which "work on substrates" must be called. In order to realize the integrated approach (and to address **P1** – **P7**), our framework includes the MiDeLoRa [4] toolkit for DL system crafting. MiDeLoRa allows for the definition of specialized provers for certain tasks, working on specialized substrates. *Provers* are conceived as regions (or single points) in the three-dimensional MiDeLoRa **space**:

Definition 2 (MiDeLoRa **Space)** *The* MiDeLoRa *space is the cartesian product* $\mathcal{S} \times \mathcal{L} \times \mathcal{T}$*, where* $\mathcal{S}$ *is the set of* substrate classes*,* $\mathcal{L}$ *is the set of supported* (DL) languages*, and* $\mathcal{T}$ *is a set of* prover tasks.

For example, $\mathcal{T}$ can contain the DL *standard inference problems* [12]: $\mathcal{T} = \{\mathsf{abox\_consistent?}, \mathsf{concept\_instances}, \ldots\}$. Again, substrates, languages and tasks are modeled as Clos classes. A MiDeLoRa *prover* is a ternary multi-method with arguments $\langle S, L, T\rangle \in \mathcal{S} \times \mathcal{L} \times \mathcal{T}$. Polymorphism is exploited for all three arguments. Since inheritance is exploited for the definitions of the classes (elements) in the sets $\mathcal{S}$, $\mathcal{L}$, and $\mathcal{T}$, a single MiDeLoRa prover defined for a point (S, L, T) can cover a whole region in the MiDeLoRa space.

2.3 Benefits of the Framework

The problems **P1** – **P7** are tackled 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., an ABox). Regarding the spatial aspects of map objects, their representation is difficult or impossible with a *standard* DL ABox (see Section 4.1). Different substrate classes thus provide different extensional representation means. Substrates can also be *hybrid* and thus allow creation of layered representations: In a hybrid substrate, a DL ABox can be combined with some other arbitrary substrate, e.g., an SBox. Thus, the DL applicability problem can be defused pragmatically.

P2, "Data and Expressivity Scalability Problem". DLs form a whole *family* of representation languages. In principle, DLs account for *expressivity scalability. Data scalability* can nowadays be achieved for simpler DLs, or RDF(S). In order to achieve data scalability, not only the knowledge level,

[4] Michael's Description Logic Reasoner

but also the symbol level must also be considered. A *(persistent) database substrate* can be used if the extensional data is extensive (substrate graphs are then stored in an RDMS). Thus, the framework accounts for data scalability. It also accounts for expressivity scalability, since MiDeLoRa allows for the definition of language-specific provers. However, the services of standard DL systems such as RacerPro are also available to the framework.

P3, "Interoperability and Middleware Problem". The substrate data model can provide an abstraction layer on top of which the OBIS is built. This abstraction layer can, for example, shield the client code of the OBIS from details in the APIs of different DL systems (see also the *Design Patterns Adapter, Bridge and Facade* in [39]). A substrate can offer caching mechanisms, abstract from remote vs. local API procedure calls, etc. A substrate *can* thus also play the role of a *mediator or semantic middleware.* The remote componentware system need not even be a DL system, but can also be an RDF(S) triple store, an RDMS on which a graph view is established, etc.

Since substrates are Clos classes utilizing inheritance which implement interfaces, additional services can easily be offered by means of substrate subclassing. For example, a RacerPro *substrate* class will offer unique RacerPro services as methods in addition to the methods that are inherited from its *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. A substrate can also be made persistent in a file or a mySQL database.

P5, "Extensibility Problem". Extensibility and openness of the architecture is obviously realized, since object-orientation supports the well-known "Open-Closed Principle". However, reuse in frameworks has been identified as problematic in some cases, because inheritance-based reuse is "white box reuse" which thus requires knowledge about the internals of the class machinery. It is known that *domain specific languages (DSLs)* can resolve some of these problems [40].

P6, "Missing QL Problem". To address this problem, the SuQL engine is provided, which is as open, extensible and parameterizable as the rest of the framework. Decidability is guaranteed given that the required entailment relationship is decidable. From a theoretician's point of view, SuQL offers unions of grounded conjunctive queries (see Section 5).

P7, "Software-Abstraction Problem". We have argued that appropri-

ate domain-specific software abstractions shall be provided in order to ensure maintainability and comprehensibility of a DL system and to avoid the "big ball of mud" (as understood in Software Engineering) syndrome in the life of a DL system.

Our approach is to define many small, comprehensible and specific provers for specific problems instead of just one big prover (implementing the core inference problem for a very expressive DL). The MiDeLoRa space provides the general structure for pinpointing provers. It provides a domain-specific software abstraction. The different provers are more comprehensible and concise than one big prover, since optimization techniques can be better localized (see Section 1.1). However, a big number of smaller provers can only be more maintainable and comprehensible if appropriate software abstractions are provided. MiDeLoRa offers *prover definition languages*, which can be understood as DSLs. Provers defined in these DSLs are almost as concise and comprehensible as the mathematical tableaux calculi used for DLs [14,30]. Due to a lack of space we cannot present the details of MiDeLoRa in this paper.

3 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).

3.1 The DISK *Data*

We are using digital vector maps of 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. 1. 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 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 prevents the intuitive usage of common

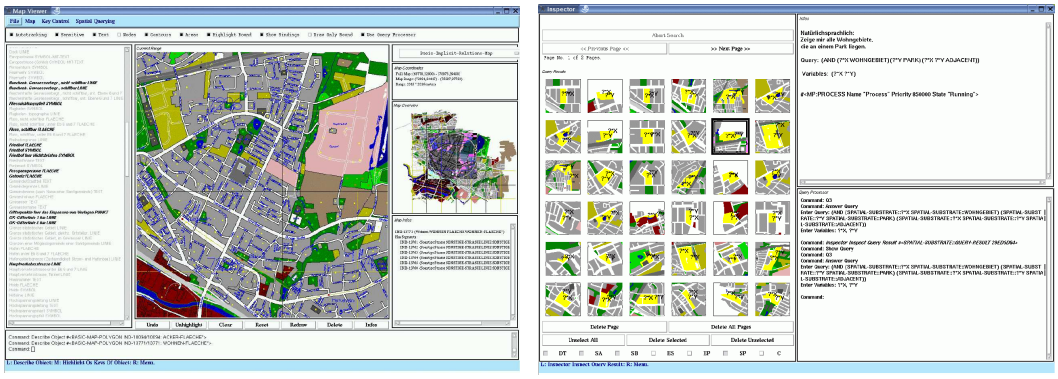

Fig. 1. The Map Viewer and Query Inspector of the DLMAPS System

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 required TA "public park containing a lake" as a "park which is public and contains a lake" with a TBox axiom such as

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

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

and we might want to retrieve the 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 *told* 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*[5] *of map objects,* a QL with variables is needed whose answer tuples can be visualized as in Fig. 1.

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 distance, etc.

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

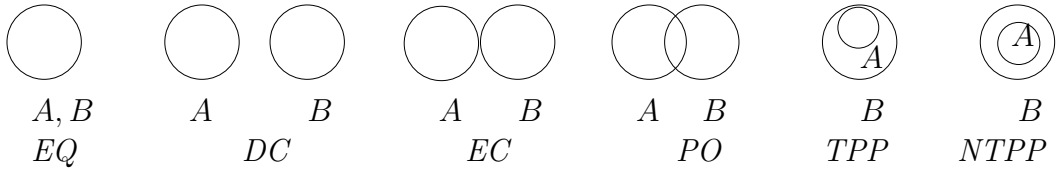

Fig. 2. RCC8 base relations: EQ = *Eq*ual, DC = *D*is*c*onnected , EC = *E*xternally *C*onnected, PO = *P*artial *O*verlap, TPP = *T*angential *P*roper *P*art, $NTPP$ = *N*on-*T*angential *P*roper *P*art. All relations with the exception of TPP and $NTPP$ are symmetric; the inverse relations of TPP and $NTPP$ are called $TPPI$ and $NTPPI$.

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*.

Thus, there are different thematic and spatial aspects one would like to represent in the extensional component and subsequently query. 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, spatial attributes such as the area or length of a map object can in principle also be derived (computed from the geometry), although this will not be very accurate. A function which exploits the map geometry to compute or verify a certain spatial aspect (for example, whether a certain qualitative relationship holds between two map objects) is called an *inspection method* in the following. This notion is defined as follows:

Definition 3 (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 [41], see Fig. 2.

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

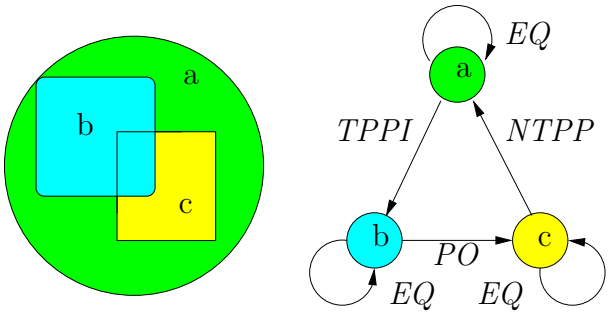

Fig. 3. Geometric constellation and its RCC network (inverse edges omitted)

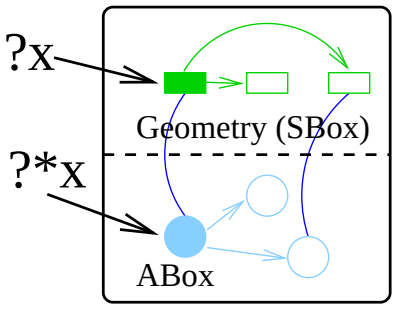

Fig. 4. Hybrid substrate: variables are bound in parallel

4 Representing and Querying the DISK

Spatial representations are, *in principle,* possible with expressive spatial *concrete domains (CDs)* [42,43] or specialized DLs [44] or spatial modal logics [45]. However, many of these logics are either undecidable, or if they are decidable, no mature DL system supporting these non-standard DLs exists. In principle, MiDeLoRa allows for the definition of tableaux provers for such specialized languages. However, in this paper we focus on more pragmatic representations which incorporate RacerPro.

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 satisfactorily 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.

4.1 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 the 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. 4(a) a "geometric scene" and its corresponding RCC8 network is depicted. Such a network will always take the form of an edge-labeled complete graph [6], due to the *JEPD property* of the RCC base relations: The base relations are *j*ointly *e*xhaustive and *p*airwise *d*isjoint). Moreover, an RCC network derived from a geometric scene will

[6] Such a graph is called a K_n in graph theory.

always be *RCC consistent* (see Section 4.4).

Moreover, 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 supports spatio-thematic concept definitions in the TBox, for example

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

($\exists contains.lake$ is short for $(\exists TPPI.lake) \sqcup (\exists NTPPI.lake)$ for reasons of readability), 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 appropriate RCC role assertions are present as well.

In principle, the specific properties of qualitative spatial (RCC) relationships cannot be captured completely within $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ as roles (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 $\mathcal{ALCI}_{\mathcal{RCC}}$ [44,46]. Even though this DL is undecidable [45], the corresponding prover has successfully computed taxonomies of $\mathcal{ALCI}_{\mathcal{RCC}8}$ 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

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

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 $n = |\{\, j \,|\, (i,j) : R \in \mathcal{A} \,\}|$ 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 $n = \{\, x \,|\, (i^{\mathcal{I}}, x) \in R^{\mathcal{I}} \,\}$; 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

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

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.

Thus, we have argued that spatio-thematic ontology-based query answering

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

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.*

4.1.1 Using an Expressive ABox Query Language

We now demonstrate that the RacerPro ABox query language nRQL [33,34] offers valuable additional query formulation facilities in this scenario. For now, we are using grounded conjunctive queries in mathematical (Horn-logic) syntax and assume that the reader has an *intuitive understanding (in addition to our explanations).* The semantics of SuQL (and nRQL) will be defined formally in Section 5 (Section 5.1). We demonstrate that nRQL's *negation as failure (NAF negation)* enables 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* provably not *adjacent to industrial areas.* Thus, all adjacent areas are provably not industrial areas (note that *adjacent* is recognized as synonym for *EC*):

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

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

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

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 known adjacent industrial areas known:*

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

The subquery $\backslash(\pi(?living\_area)\ \ adjacent(?living\_area, ?i), ind.\_area(?i))$ returns the complement set of the answer to the query $ans(?living\_area) \leftarrow adjacent(?living\_area, ?i), ind.\_area(?i))$.[8] So, an instance is in $\backslash(\pi(?living\_area)\ \ adjacent(?living\_area, ?i)$, $ind.\_area(?i)$ iff for $?living\_area$ there is no known adjacent industrial area present. However, in principle $?living\_area$ might have an *unknown* adjacent industrial area (in case there is no corresponding ABox individual) – thus, this query returns a *superset* of $\backslash(\exists adjacent.industrial\_area(?living\_area))$, and the query is therefore *more general* than (2).

4.1.2 Drawbacks of the ABox Representation

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

(1) The size of the generated ABoxes is significant. Since the RCC network is explicitly 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 for queries which mention spatial aspects, since spatial index structures are missing.
(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 geometric 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*

[8] Please note that π is called the *body projection operator,* see Section 5.

Domain Assumption (ODA) is made in DLs. This will be illustrated in Section 4.3.

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*. We thus switch to a hybrid representation incorporating an SBox.

4.2 Representation Option 2 – Use a Map Substrate:

Due to the problems with spatio-thematic concepts and since closed domain reasoning is all that we can achieve here anyway, 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 have 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. 4(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* and a *map substrate* as follows:

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

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, also offering *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. The SBox provides a spatial index, supporting the efficient evaluation of inspection methods by means of spatial selection operations. Computed spa-

tial aspects can also be materialized in order to avoid repeated re-computation (e.g., 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 binding is also depicted in Fig. 4.

Assume we are using a map substrate for the DISK representation. Let us consider the example query given in Section 4.1.1 again. Since the RCC network is now no longer represented in the ABox, the SBox must be queried for spatial relationships. Queries (1) and (2) from Section 4.1.1 thus no longer work.

However, query (3) 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:

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

Thus, we not only gain, but also lose something here (queries (1) and (2) cannot be expressed). This is an important insight. On the positive side, we are now able to define and evaluate spatial predicates which are richer than RCC predicates, since the geometry of the map is represented. We can thus design dedicated spatial query atoms. These spatial atoms (e.g., distance query atoms) are discussed in Section 5.2.

4.3 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 Section 2.1 that MiDeLoRa offers so-called *spatial ABoxes*. There is then 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 exploit the spatial aspects of the nodes.

With a spatial ABox, the RCC role assertions need 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. Thus, there is no need to explicitly store an $|V|^2$ number of RCC role assertions in the ABox, as they are "intrinsically represented". However, this requires a dedicated prover which can be defined in MiDeLoRa.

In Section 4.1 we have closed the ABox w.r.t. the RCC role assertions. 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 which reference an RCC role. This forces RacerPro into a *closed domain reasoning mode*; however, this is a *two-step process in the* $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ *tableau calculus.* 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). However, this is a *highly non-deterministic process.* Thus we stated in Section 4.1.2 (5) that DL reasoners are not very good at closed domain reasoning.

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 first the *reuse of an existing successor* is tried before a fresh successor is generated for an RCC role. (However, the generating rules become non-deterministic with that modification). The tableaux rules of MiDeLoRa can be parameterized to work in such a way.

4.4 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), 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.* Unlike the $\models$ relation for the SBox, which only exploits spatial model checking

by means of inspection methods, spatial inference is thus required here. The RCC substrate is, on the one hand, more expressive then the SBox, since also vague or unknown RCC relations can be expressed. On the other hand, the geometry of the map cannot be preserved (as in Option 1).

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 Section 5.3). Unlike for the map substrate, the ABox is the primary substrate S_1, since the RCC substrate is an "add on" from the perspective of the RacerPro user. Let us describe the RCC substrate:

Definition 5 *Let* $\mathcal{R} =_{def} \{EQ, DC, EC, PO, TPP, TPPI, NTPP, NTPPI\}$ *be the set of RCC8 base relations. An* RCC substrate $\mathcal{RCC}$ *is a substrate such that* V *is a set of RCC nodes with* $\mathcal{L}_\mathcal{V} = \emptyset$*, and* $\mathcal{L}_\mathcal{E} = 2^\mathcal{R}$*.*

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 (where 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 17) 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 denoting 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 substrate $\mathcal{RCC}$ *containing only base relations* can be viewed as a set of FOPL ground atoms. Such a RCC network is said to be *relationally consistent* iff $\mathcal{RCC}'$ is satisfiable:

$$\begin{aligned}\mathcal{RCC}' = \; & \mathcal{RCC} \;\cup\; \{\forall x.EQ(x,x)\} \cup \\ & \{\,\forall x,y,z.R(x,y) \wedge S(y,z) \rightarrow T_1(x,z) \vee \cdots \vee T_n(x,z) \;\mid \\ & \qquad\qquad R \circ S = \{T_1, \ldots, T_n\} \in \mathcal{RCC}_\mathcal{T} \,\} \cup \\ & \{\forall x,y. \textstyle\bigvee_{R \in \mathcal{R}} R(x,y)\} \;\cup\; \{\forall x,y. \textstyle\bigvee_{R,S \in \mathcal{R}, R \neq S} R(x,y) \wedge \neg S(x,y)\}\end{aligned}$$

For example, the network $\mathcal{RCC} = \{NTPPI(a,b), DC(b,c), PO(a,c)\}$ 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 $NTPPI \circ DC = \{DC\}$. Thus, $\mathcal{RCC}' \models DC(a,c)$, which contradicts $PO(a,c)$, due to the JEPD property.

Let us briefly define some more notions. *Entailment of RCC relations* or

RCC ground query atoms can be *reduced to inconsistency checking* as follows: $\mathcal{RCC}' \models R(a,b)$ iff $\mathcal{RCC}' \cup \{(\mathcal{R} \setminus R)(a,b)\}$ 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 $\mathcal{RCC}' \cup \{EQ, EC, PO, TPP, TPPI, NTPP, NTPPI\}(a,c)$ 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

$$\mathcal{A} = \{hamburg : german\_city, paris : french\_city, fr : country, ger : country\},$$

$$\mathcal{RCC} = \{NTPP(*hamburg, *ger), EC(*ger, *fr), NTPP(*paris, *fr)\}$$

and with the obvious mapping $*(x) = *x$ for $x \in \{hamburg, paris, fr, ger\}$. 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 present in $\mathcal{RCC}$.

5 SuQL – The Substrate Query Language Framework

In the following we describe the core design principles underlying the generic substrate query language SuQL, its instantiations (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 outlined, and additionally some examples for queries using abstract Horn-logic syntax have been given. In the following, we will use the concrete syntax of the query language framework in order to make it less abstract.[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))).
```

[9] The prefix Lisp syntax is as readable and as formal as the mathematical syntax.

The expression `(?x ?y)` is called the *head,* and `(and (?x woman) (?x ?y has-child))` the *body* of the query. 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 (generic) body constructors `and, or, neg` and `project-to`; `neg` corresponds to "\", and `project-to` to "π", as already used and briefly discussed in Section 4.1.1.

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:

Variables and individuals can be used in query atoms. Both variables and individuals are called *objects*. The variables range over V, the nodes of the substrate. Thus, SuQL offers only so-called *distinguished* or *must-bind variables* [27]. Variables are bound to nodes which *satisfy* the query – a variable binding satisfies a query iff the ground query – that is obtained from 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 $S \models P(x)_{x \leftarrow i}$. 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 the example 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.

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. 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.

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:

$$V = P(?x)^{\mathcal{E}} \cup (\backslash P(?x))^{\mathcal{E}}$$

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

Let us consider the ABox query language case again. We would like to

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

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 before the complement set is computed with `neg`. It allows to "fold in" subbodies for which dedicated horn rules would have to be written otherwise. For example, in order to retrieve those individuals which do *not* have a *known* child, we have to use `(neg (project-to (?x) (?x ?y has-child)))`, since the extension of `(neg (?x ?y has-child))` is a *two-dimensional set*.

5.1 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 grammar (*{a|b} *means* `a` *or* `b`*):*

$$
\begin{aligned}
\textit{head} &:= \texttt{(}\textit{object}^{*}\texttt{)} \\
\textit{object} &:= \textit{variable} \mid \textit{individual} \\
\textit{variable} &:= \textit{a symbol beginning with } \texttt{?} \\
\textit{individual} &:= \textit{a symbol} \\
\textit{body} &:= \textit{atom} \mid \texttt{(} \{\texttt{and} \mid \texttt{union}\}\ \textit{body}^{*}\texttt{)} \mid \texttt{(neg}\ \textit{body}\texttt{)} \mid \\
&\quad\ \texttt{(project-to (}\textit{object}^{*}\texttt{)}\ \textit{body}\texttt{)} \\
\textit{atom} &:= \textit{unary\_atom} \mid \textit{binary\_atom} \mid \textit{equality\_atom} \\
\textit{unary\_atom} &:= \texttt{(}\textit{object unary\_atom\_predicate}\texttt{)} \\
\textit{binary\_atom} &:= \texttt{(}\textit{object object binary\_atom\_predicate}\texttt{)} \\
\textit{equality\_atom} &:= \texttt{(}\textit{object object}\ \{=_{syn} \mid =_{sem}\}\texttt{)}
\end{aligned}
$$

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.

[10] The predicate $=_{sem}$ is the standard equality predicate in FOPL with equality.

The function $\mathsf{obs}(q)$ *returns the* objects (individuals and variables) *referenced in* q *and 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* $Q \in \{binary\_atom\_predicate, =_{syn}, =_{sem}\}$, $\mathsf{obs}((\{\ \texttt{and}\ |\ \texttt{union}\ |\ \texttt{neg}\ \}\ q_1 \ldots q_m)) =_{def} \bigcup_{1 \leq i \leq m} \mathsf{obs}(q_i)$, *but* $\mathsf{obs}((\texttt{project-to}\ (x_1 \ldots x_m) \ldots)) =_{def} \{x_1 \ldots x_m\}$. *Thus,* obs *"stops at projections".*

Before we can define the semantics we need 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 1s, 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

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

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 1s 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 $\mathcal{B}$.

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.*

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 equality atoms for query atoms which reference individuals. The query body q *is thus first rewritten. We define* $\Theta(q)$ *for 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 extend 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 some fixed enumeration of $\mathsf{obs}(q')$ (so $n = |\mathsf{obs}(q')|$).

We define $.^{\mathcal{E}}$ 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 to Θ, all unary and binary query atoms which are not equality atoms now have one and two variables correspondingly). The semantics of the equality predicates is fixed as follows: $S \models i =_{syn} i$ and $S \not\models i =_{syn} j$, and $S \models i =_{sem} j$ 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 $.^{\mathcal{E}}$ inductively for complex (sub)bodies in q':

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

To get the final answer of a query, the head *has to be considered, for a final projection. Thus, the result of* `(retrieve` *head* `q)` *is simply given as*

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

5.2 The nRQL *Instantiation of the* SuQL

nRQL [33,34] is a specialized SuQL. It offers dedicated query atoms for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, e.g. atoms addressing the concrete domain part of an ABox. The nRQL atoms are: *concept query atoms*, e.g. `(?x (some has-child human))`;

role query atoms, e.g. `(?x ?y has-child)`, and (binary) *constraint query atoms*. All atoms have been discussed already, with the exception of constraint query atoms. The following query uses all kinds of nRQL 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)`.[11] However, these "values" are in fact variables in a concrete domain constraint network (which can be left unspecified, i.e., no syntactically specified so-called *told value* must exist).

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 $\{\texttt{not}, \texttt{inv}\}$. Thus, nRQL offers not only NAF negated roles, but also *classical negated roles*, which are not provided by $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$.

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 [12]; e.g., for a concept query atom predicate $P = C$ this boils down to ordinary instance checking or an 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 one of the RacerPro API functions concept_instances or individual_instance? need 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, since negated roles are not supported in $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$: $\mathcal{A} \models \Phi(\neg R)_{x \leftarrow i, y \leftarrow j}$ iff $\mathcal{A} \cup \{(i, j) : R\}$ is unsatisfiable. These "reduction tricks" are well known

[11] 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 where the two chains end in different attributes).

[47]. Particular API functions are called for constraint query atoms as well.

5.3 Concrete SuQL *Instantiations for the* DLMAPS *System*

We have discussed four representation options in the DLMAPS system. Although the principal ideas have been laid out, we briefly present 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?

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

Option 2: The resulting hybrid QL is called SnRQL. It provides the following additional spatial atoms (note that it 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 inspection methods.

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 (or ∞); 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 than 100 meters from i.* Either the shortest distance or the distance between the centroids of these objects is used.

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 a size specified by the interval $[min; max]$ 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 type polygon in V whose area is in $[100; 1000]$. Also `:length` is understood for linear objects. 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). Here is a 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 the queries are no longer hybrid. 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. The same syntax is used as for the SnRQL RCC atoms (but the implementation obviously differs, since geometric computations are required in one case, and RCC constraint checking in the other case).

6 The SuQL Query Answering Engine

The SuQL engine exploits two generic optimization techniques: a cost-based syntactic optimizer [34] and a semantics-based optimization, the so-called *query repository* [34,48] (which is not described here).

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 order 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 for, which can be overridden for specialized atoms. A plan is thus generated step-by-step, and given a plan $(a_1, \ldots, a_n)$, an atom a_{n+1} which yields the maximal score is selected and added next to the plan: $(a_1, \ldots, a_n, a_{n+1})$. In such a way, a heuristic search procedure using standard *beam search (of sufficient breadth)* searches for a "good" plan (if n gets big, not all of the $n!$ plans can be considered).

The standard implementation of `get-score` simply considers the *role* of the atom in the currently evaluated plan and weights the atom accordingly. A unary atom can either play the role of a *tester* or of a *generator*, depending on

whether the variable referenced in the atom will be already bound at execution time (specified by the plan) or not. The standard `get-score` implementation simply prefers *testers over generators*. A binary atom `(?x ?y R)` can additionally take the role of a *successor generator* where only `?x` is already bound, or of a *predecessor generator* in case only `?y` is already bound.

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 C)` 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 `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) will be preferred otherwise. Since this information is not always available, one sometimes has to rely on *told information.*

We have performed an evaluation of the effectiveness of this optimization technique, using the so-called *Lehigh University Benchmark (LUBM)* and the LUBM Query No. 9 (**Q9** in the following) [49]. The details of the LUBM are not important here. It is sufficient to state that the LUBM consists of an ontology modeling the university domain, as well as an ABox containing thousands of individuals. The size of this ABox can be scaled, depending on the number of *Universities* and *University Departments*. **Q9** is the following query:

```
(retrieve (?x ?y ?z)
   (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*. The rationale for doing so is to 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. The fastest

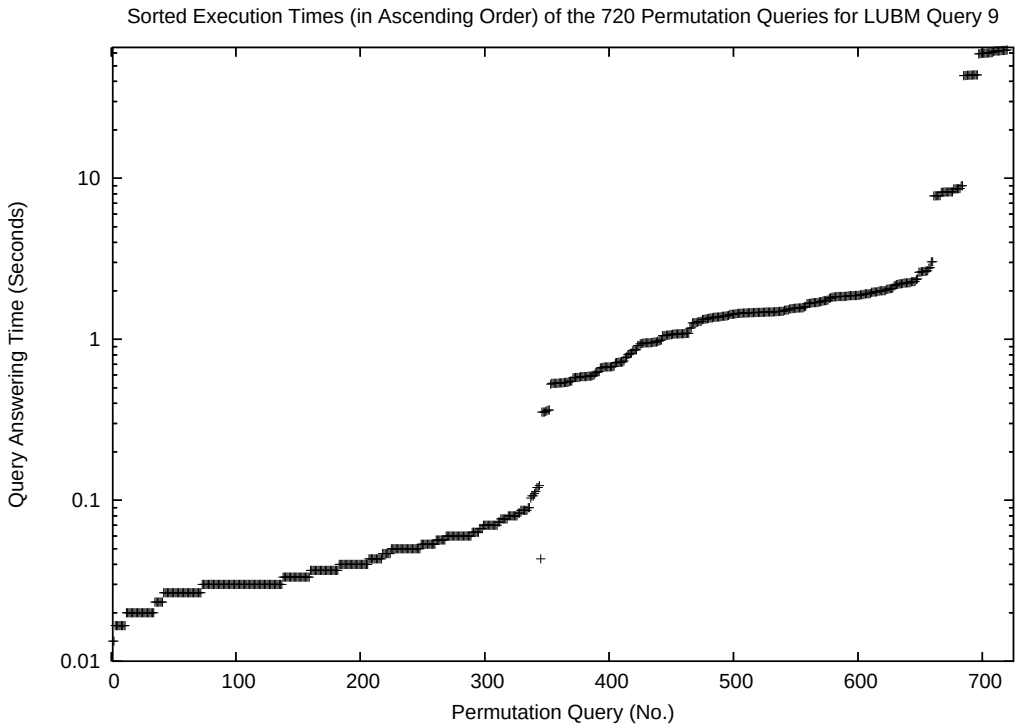

Fig. 5. Execution Times of the 720 Permutation Queries for **Q9**

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. We did not measure variances. The result shows that this optimization is *good enough* and indispensable.

Another very important optimization technique exploited concerns the computation of the *role successors and role predecessors*; this technique is very important, since the optimizer prefers a navigational approach to variable binding computation (since successor 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:

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

This Lemma is a direct consequence of [15, pp. 11, Def. 9,6.]; the optimization is also suggested in [27] and [12, pp. 67]. We omit the (rather trivial) proof here. The Lemma has been exploited for nRQL starting with RacerPro 1.8.0 (even for the so-called incomplete modi which do not require ABox reasoning [48]). Please note that the Lemma no longer holds as soon as equality statements (or `owl:same-as`), functional roles or number restrictions are added. However, a weaker statement can then 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 or same-as assertions in the ABox between i and j. The test can be used as a *guard* for the ABox satisfiability tests which reduces the number of tests.

Let us use **Q9** to demonstrate the effectiveness of the optimizations. 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 does not yet achieve the performance of RacerPro). The initial ABox satisfiability test (which is needed in order to decide whether query answering makes sense at all) still takes 12 seconds, but the subsequent ABox satisfiability checks run faster due to optimizations described in [29,21]. Such a subsequent ABox test currently needs 3.7 seconds. For answering **Q9**, not a single ABox consistency test was performed. We have counted the number of calls to the API function individuals_related?, initiated by the role tester, role successor and role predecessor generator functions. Given individuals i, j, a role R and an ABox $\mathcal{A}$ (with associated TBox), individuals_related?$(\mathcal{A}, i, j, R,)$ decides $\mathcal{A} \models R(i, j)$. individuals_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 individuals_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 naïve implementation would perhaps even generate $1555^2 = 2418025$ ABox consistency checks, thus, 103.37 days would be needed. These numbers demonstrate the significance of the optimizations.

7 Conclusion

Building OBIS with enabling DL technology is a non-trivial task, especially for IS in non-standard domains such as the one considered here. The space of design decisions is very large. Thus we have designed a flexible and generic framework which offers appropriate abstractions that are able to cover *regions* in these design spaces instead of just points.

Since decidability and scalability is not always easy to achieve for OBIS, we believe that it is of even more importance to identify practical solutions which, even though they do not exploit or advance the latest theoretical state-of-the-art techniques in DL research, can nevertheless be considered an advance regarding the current state-of-the-art IS technology and provide guidance and "road maps" for similar designs.

We claim that our framework for building pragmatic combinations of specialized representation layers (including DL ABoxes) for which orthogonal specialized substrate QLs and dedicated provers can be defined, provides a great deal of flexibility for building similar OBIS. Moreover, some of the functionality described here is immediately available for other users (of the RacerPro system). We claim that the identified software abstractions are valuable, also for non-Lisp developers, as are the identified optimization techniques for ontology query answering engines.

8 Acknowledgments

We want to thank the anonymous reviewers of this paper, our users, as well as Volker Haarslev, Kay Hidde and Mike Fischer for their very valuable and thoughtful comments. We would also like to thank the land surveying office of the city of Hamburg (Landesbetrieb Geoinformation und Vermessung Hamburg) for providing us with the DISK data.

References

[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. Conference on the Principles of Knowledge Representation and Reasoning (KR'98), 1998.

[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, Vögele, T. Visser, U. Stuckenschmidt, H. Schuster, G. Neumann, S. Hübner, Ontology-based Integration of Information - A Survey of Existing Approaches, in: Proc. IJCAI-01 Workshop on Ontologies and Information Sharing, 2001.

[6] M. Wessel, Some Practical Issues in Building a Hybrid Deductive Geographic Information System with a DL Component, in: Proc. of the 10th Int. Workshop on Knowledge Representation Meets Databases (KRDB'03), 2003.

[7] M. Wessel, R. Möller, A Flexible DL-based Architecture for Deductive Information Systems, in: Proc. of the IJCAR-06 Workshop on Empirically Successful Computerized Reasoning (ESCOR), 2006.

[8] R. Ferber, Information Retrieval, dpunkt.verlag, 2003.

[9] I. Horrocks, D. L. McGuinness, C. A. Welty, Digital Libraries and Web-based Information Systems, 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.

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

[13] V. Haarslev, R. Möller, RACER System Description, in: Proc. of the Int. Joint Conference on Automated Reasoning (IJCAI'01), 2001.

[14] I. Horrocks, U. Sattler, S. Tobies, Practical Reasoning for Expressive Description Logics, in: Proc. of the 6th Int. Conference on Logic for Programming and Automated Reasoning (LPAR'99), 1999.

[15] V. Haarslev, R. Möller, M. Wessel, The Description Logic $\mathcal{ALCNH}_{\mathcal{R}^+}$ Extended with Concrete Domains: A Practically Motivated Approach, in: Proc. of the Int. Joint Conference on Automated Reasoning (IJCAR'01), 2001.

[16] D. Calvanese, G. De Giacomo, M. Lenzerini, What can Knowledge Representation do for Semi-structured Data?, in: Proc. of the 15th National Conference on Artificial Intelligence (AAAI'98), 1998.

[17] F. Manola, E. Miller, RDF Primer, Tech. Rep., World Wide Web Consortium (Feb. 2004).

[18] D. Brickley, R. V. Guha, Resource Description Framework (RDF) Schema Specification 1.0, Tech. Rep., World Wide Web Consortium (Mar. 2000).

[19] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, DL-Lite: Tractable Description Logics for Ontologies, in: Proc. of the 20th National Conference on Artificial Intelligence (AAAI'05), 2005.

[20] V. Haarslev, R. Möller, M. Wessel, Description Logic Inference Technology: Lessions Learned in the Trenches, in: Proc. of the Int. Workshop on Description Logics (DL'05), 2005.

[21] R. Möller, V. Haarslev, M. Wessel, On the Scalability of Description Logic Instance Retrieval, in: Deutsche Jahrestagung für Künstliche Intelligenz (KI'06), 2006.

[22] E. Karabaev, C. Lutz, MONA as a DL Reasoner, in: Proc. of the Int. Workshop on Description Logics (DL'05), 2004.

[23] S. Bechhofer, R. Möller, P. Crowther, The DIG Description Logic Interface, in: Proc. of the Int. Workshop on Description Logics (DL'03), 2003.

[24] 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: Proc. of the Int. Workshop on OWL: Experiences and Directions (OWLED'06), 2006.

[25] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, AL-Log: Integrating Datalog and Description Logics, Journal of Intelligent Information Systems 10 (3), 1998.

[26] A. Y. Levy, M.-C. Rousset, CARIN: A Representation Language Combining Horn Rules and Description Logics, in: European Conference on Artificial Intelligence (ECAI'96), 1996.

[27] I. Horrocks, S. Tessaris, Querying the Semantic Web: a Formal Approach, in: Proc. of the 2002 Int. Semantic Web Conference (ISWC'02), 2002.

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

[29] V. Haarslev, R. Möller, Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results, in: Ninth Int. Conference on the Principles of Knowledge Representation and Reasoning (KR'04), 2004.

[30] F. Baader, U. Sattler, Tableau Algorithms for Description Logics, in: Proc. of the 9th Int. Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'00), 2000.

[31] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, R. Rosati, QuOnto: QUerying ONTOlogies, in: Proc. of the 20th National Conference on Artificial Intelligence (AAAI'05), 2005.

[32] A. Newell, The Knowledge Level, Artificial Intelligence 18 (1) (1982) 87–127.

[33] V. Haarslev, R. Möller, M. Wessel, Querying the Semantic Web with Racer + nRQL, in: Proc. of the Int. Description Logic Workshop (DL'04), 2004.

[34] V. Haarslev, R. Möller, M. Wessel, A High Performance Semantic Web Query Answering Engine, in: Proc. of the Int. Description Logic Workshop (DL'05), 2005.

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

[36] B. McBride, Jena: A Semantic Web Toolkit, IEEE Internet Computing 6 (6) (2002).

[37] K. Clark, Negation as Failure Logic and Databases, in: H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum, 1978.

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

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

[40] 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.

[41] D. A. Randell, Z. Cui, A. G. Cohn, A Spatial Logic Based on Regions and Connections, in: Proc. of the Int. Conference on Principles of Knowledge Representation and Reasoning (KR'92), 1992.

[42] V. Haarslev, C. Lutz, R. Möller, A Description Logic with Concrete Domains and Role-Forming Predicates, Journal of Logic and Computation 9 (3) (1999).

[43] C. Lutz, M. Milicic, A Tableau Algorithm for DLs with Concrete Domains and GCIs, in: Proc. of the Int. Workshop on Description Logics (DL'05), 2005.

[44] 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, 2003.

[45] C. Lutz, F. Wolter, Modal Logics of Topological Relations, in: Proc. of Advances in Modal Logics (AIML'04), 2004.

[46] M. Wessel, On Spatial Reasoning with Description Logics - Position Paper, in: Proc. of the Int. Workshop on Description Logics (DL'02), 2002.

[47] R. Möller, M. Wessel, Terminological Default Reasoning About Spatial Information: A First Step, in: Proc. of the Int. Conference on Spatial Information Theory (COSIT'99), 1999.

[48] Racer Systems GmbH & Co. KG, RacerPro User's Guide 1.9.0, Tech. Rep., `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf`, 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 Conference (ISWC'04), 2004.