A Flexible DL-based Architecture for Deductive Information Systems

Michael Wessel, Ralf Möller
Hamburg University of Technology (TUHH), Germany
{mi.wessel, r.f.moeller}@tuhh.de

1 Introduction

Description Logics (DLs) are nowadays an accepted standard for decidable knowledge representation. DLs also provide the theoretical foundation for representing and reasoning with ontologies as well as for the Semantic Web. Thus, DL systems (e.g., Fact++, Pellet, RacerPro) and DL-based technology in general will play an increasingly important role for building the next generation of deductive, ontology-based information systems.

The RacerPro description logic system [HM01a] implements the very expressive DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, also known as $\mathcal{SHIQ}(D^-)$ [HST99, HM01b]. RacerPro's most prominent feature is the support for so-called *ABoxes* which allow for the representation of a "concrete state of the world" in terms of individuals and relationships. In the following, we assume familiarity with DLs as well as DL systems.

In order to provide the information technology backbone for next generation information systems (IS), current DL systems still have a long way to go. Nowadays, building ontology-based IS with a DL system is still a non-trivial task: One the one hand, DLs somehow live in their own "realm" and are thus not really interoperable with the rest of "more conventional" IS infrastructure (e.g., existing relational databases). One the other hand, only recently issues such as persistency and powerful *query languages (QLs)* have been considered and incorporated into DL systems. DLs itself have their deficiencies as well and are thus not a panacea for arbitrary information representation and (ontology-based) retrieval tasks: Due to the complexity of the inference problems, scalability (in the average case) is not easy to achieve and nowadays only achievable for *knowledge bases (KBs)* which use simple DLs. Standard DLs are well suited for the representation of (and reasoning about) semi-structured (or even unknown/indeterminate) information, but things become more complicated if, say, n-ary relationships or special "non abstract" domains such as space are considered. Then, either non-standard DLs or complicated logical encodings are needed. For non-standard DLs, no working systems exist, and complicated logical encodings are likely to decrease the performance and complicate the information handling and maintenance.

To get working systems on time *today*, we must thus agree upon pragmatic solutions. This also implies that existing technology (i.e., existing DL system such as RacerPro) must be exploited and possibly extended for the task of IS building. Due to the intellectual inherent complexity of the field, application and system studies on how to use DL system in real-world IS scenarios are thus of utmost importance in order to provide guidance.

In this paper, we consider the IS domain of deductive geographic information systems (GIS), which can be called a "non-standard domain". We describe the problems encountered and pragmatic solutions found during the endeavor of *building an ontology-based query answering system for digital city maps*, the DLMAPS system [Wes03a]. We use RacerPro as a DL system, which can be called an empirically successful system, since it is widely used.[1] In this IS domain of digital city maps, we must

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

The paper provides the following contributions:

We describe how an existing DL system such as RacerPro can be used for building an ontology-based IS in such a "non-standard" IS domain for which a standard DL such as $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ is not very well suited. This mainly concerns the representation of the spatial aspects of the maps, which we call the "spatial representation problem" in the following. We will demonstrate which representation options are available in this setting.

We demonstrate what can be done with nowadays available state-of-the art DL (and Semantic Web) technology such as RacerPro, and identify and motivate and design pragmatic extensions to tackle the spatial representation problem. Making this explicit by means of this paper will help other users to build their own IS with enabling DL technology by exploiting similar "design patterns" as we did.

Spatial representations are, in principle, possible with expressive spatial *concrete domains (CDs)* [HLM99, LM05] or specialized DLs [Wes03b] or spatial modal logics [LW04]. However, no optimized mature DL system supporting these non-standard DLs exist, and building a DL system which supports them is a non-trivial task.

Even if the spatial representation is achieved, then additionally an expressive spatio-thematic QL is needed for the IS. Designing and implementing such a QL is again a non-trivial task. We will demonstrate (from a pragmatic perspective) how such a QL can be designed. Moreover, the QL is also implemented and available for other users.

RacerPro has been extended in 2003 by an expressive QL called nRQL [WM05, KG06]. Given that we have solved the spatial representation problem, we can use and extend nRQL by "spatial atoms" to get the desired spatio-thematic query language. The nRQL query answering engine (which is an integral part of RacerPro) can be called an empirically successful system, since it is used by many RacerPro users. At the time of this writing, this engine is (to the best of our knowledge) the only optimized ABox query answering engine which provides complex ABox queries and also addresses issues such as life cycle management of queries (queries are managed as objects which have a state), concurrent and incremental query answering, version control of query answers, defined queries, etc.

Another contribution of this paper is the identification and description of (software) abstractions which we claim can be useful for other developers of ontology-based IS with

[1] And is commercially distributed by the start-up company Racer Systems.

DL system components as well: We have based the DLMAPS system on the so-called *substrate data model.* This data model serves as a *layer of indirection and abstraction*, shielding the IS from the details of the used DL system (a kind of "semantic middle ware"). But more importantly, the substrate data model enables us to attach or associate additional information on or with ABox individuals, for which no encoding in an ABox is possible or appropriate. We will show how the substrate data model can be used to solve the spatial representation problem of the map. The resulting representations will by hybrid; thus, a *hybrid QL* will be needed to combine the information distributed on the different substrate layers.

The paper is structured as follows. We first describe the IS domain of digital city maps, the concrete map data we use, and the idea of ontology-based queries to such city maps. We then present various options how to represent the maps. Next we describe the nRQL ABox QL which plays a crucial role here, since nRQL is the QL which is extended to become a hybrid spatio-thematic QL in this paper. Finally, the resulting QL is presented. Then comes the conclusion.

2 The Scenario - Ontology-Based Queries To a City Map

For what follows, we call the TBox or ontology of the DLMAPS IS the *intensional component,* whereas the actual map data is kept in the *extensional component. Ontology-based query answering* then means that the (defined) vocabulary from the intensional component can be used in queries to retrieve the desired information (by means of query answers) from the extensional component. The *query answering component* is responsible for computing these answers.

The data in the DLMAPS IS are digital vector maps from the city of Hamburg provided by the land surveying office ("Amt für Geoinformation und Vermessungswesen Hamburg"); these maps are called the DISK ("Digitale Stadtkarte"). Part of the DISK is visualized by the Map Viewer component of our system in Fig. 1. Each map object (also called *geographic feature*) is *thematically annotated*. The basic *thematic annotation (TA)* has been established by the land surveying office itself. The TA says something about the "theme" or semantics of the map object. Simple symbols/names such as "green area", "meadow", "public park", "lake" are used. A few hundred TAs are used and documented in a so-called *thematic dictionary (TD)*, which is GIS-typically organized in 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 such 'as "cemetery" is often missing. This is unfortunate, since it prevents the usage of common sense vocabulary for query formulation. We can repair this defect by adding a background ontology (in the form of a TBox) providing generalizing TAs by means of taxonomic relationships. On the other hand, *defined concepts* ("if and only if") can be added and exploited to *automatically enrich* the given basic annotations. Thus, we might define our own needed TA "public park containing a lake" as a "park which is public which contains a lake" with a TBox axioms such as

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

or

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

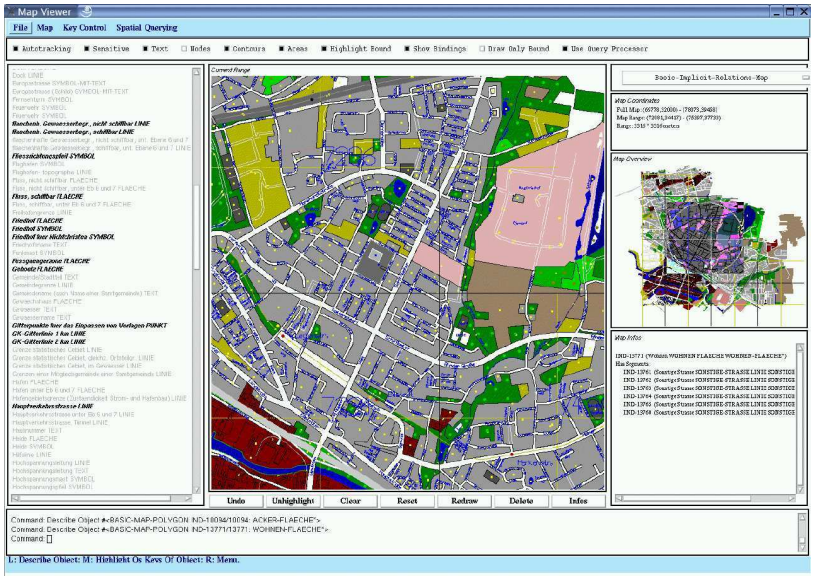

Figure 1: The Map Viewer of the DLMAPS System

and we might want to retrieve all instances of these concepts. This is what ontology based query answering is all about. Inference is required to obtain these instances, since there are no *known* instances of *public_park_containing_a_lake* (this is not among the basic TAs provided by the land surveying office). For simple queries, *instance retrieval queries* might be sufficient as a QL. However, in our scenario one must retrieve *constellations*[2] *of map objects* which satisfy a *complex query expression;* thus, a QL with variables is needed.

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

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

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

We use the following terminology: a *thematic concept* refers only to thematic aspects, similarly a *spatial concept* solely to spatial aspects, whereas a *spatio-thematic concept* refers to both. We also talk of *thematic*, *spatial* and *spatio-thematic queries*. A strict separation might be difficult sometimes.

Thus, there are different thematic and spatial aspects one would like to represent and address with the spatial QL. Since the concrete geometry is given by the map, the spatial aspects of the map objects are in principle intrinsically represented and available. This mainly concerns the spatial relationships which are depicted in the map. However, also attributes like the area etc. can in principle be computed from the geometry (although this will not be very accurate). A function that exploits the geometry to dynamically check the requested spatial aspect (i.e., spatial property or

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

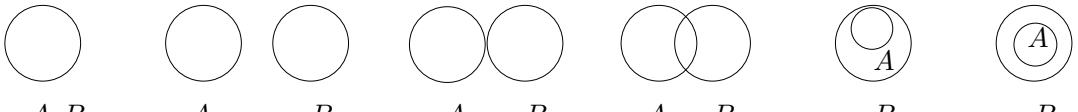

Figure 2: RCC8 base relations: EQ = Equal, DC = Disconnected , EC = Externally Connected, PO = Partial Overlap, TPP = Tangential Proper Part, $NTPP$ = Non-Tangential Proper Part. All relations with the exception of TPP and $NTPP$ are symmetric; the inverse relations of TPP and $NTPP$ are called $TPPI$ and $NTPPI$.

relation) is called an *inspection method* in the following.

It is obvious that *qualitative* spatial descriptions are of great importance. On the one hand, they are needed in order to be able to define concepts in the TBox such as "public park containing a lake". On the other hand, they are needed in the spatial QL ("retrieve all public parks containing a lake"). A popular and well-known set of qualitative spatial relationships is given by the RCC8 relations, see Fig. 2.

One 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. However, if we want to use a RacerPro ABox for the extensional component, then the spatial representation options are limited, and we must necessarily make use qualitative descriptions for the map representation, see below.

The DLMAPS system will support ontology-based spatio-thematic conjunctive queries to the DISK. To give a first impression of what will be possible, suppose we want to identify the chemical plants in the DISK which might be responsible for a chemically contaminated lake:

$$\begin{aligned} ans(?lake, ?park, ?creek, ?industrial\_area, ?chemical\_plant) \leftarrow \\ lake(?lake), chemically\_contaminated(?lake), \\ park(?park), public(?park), contains(?park, ?lake), \\ creek(?creek), flows\_in(?creek, ?lake), \\ crosses(?creek, ?industrial\_area), \\ contains(?industrial\_area, ?chemical\_plant), unreliable(?chemical\_plant). \end{aligned}$$

We assume that the reader has an intuitive understanding of such a query. The different conjuncts of such a query are called *query atoms* in the following. A *ground query atom* is an atom in which the free variables have been substituted with (satisfying) individuals resp. "constants". We formalize these notions subsequently. An atom with one variable is equivalent to a simple *instance retrieval query*.

3 Representing and Querying the DISK

It is clear that the kind of representation we will devise for the DISK in the extensional component will also determine what we can query, and how we can query (i.e., which techniques to be used for computing the query answers). Without doubt, the thematic aspects of the DISK map objects can be represented satisfactory with a standard DL.

To solve the spatial representation problem of the DISK in the extensional component, we consider three representation options and analyze their impacts:

• **Representation Option 1 – Use a Single ABox:** We can represent "as much spatial aspects as possible" in a $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ ABox. Regarding the spatial

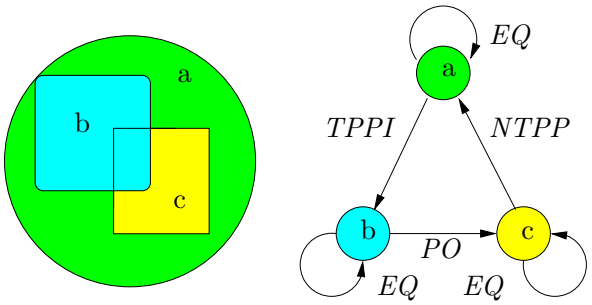

Figure 3(a): Concrete geometric scene and corresponding RCC8 network (inverse edges omitted)

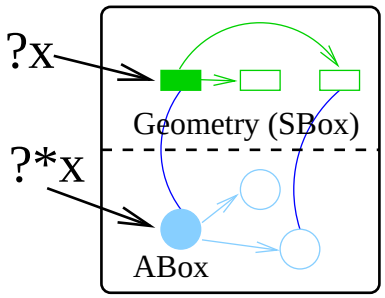

Figure 3(b): Hybrid map substrate: variables are bound in parallel

relationships, we can only represent *qualitative* relationships. From the concrete map geometry, we can compute a so-called *RCC network* and represent this by means of *RCC role assertions* in the ABox, e.g. $(i, j) : TPPI$ etc. In Abb. 3(a) a "scene" and its corresponding RCC8 network is depicted. Such a network will always take the form of a an edge labeled complete graph, a so-called K_n (see graph theory), due to the *JEPD property* of the RCC base relations: The base relations are *j*ointly *e*xhaustive, *p*airwise *d*isjoint). Moreover, an RCC network derived from a geometric scene will always be *RCC consistent* (see below).

Moreover, selected spatial attributes such as area and length can be represented in the CD of the ABox; we then use *concept assertions* such as $i : \exists(has\_area).=_{12.345}$.

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

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

(we are using $\exists contains.lake$ instead of $(\exists TPPI.lake) \sqcup (\exists NTPPI.lake)$). Obviously, an individual i in the ABox can only be recognized as an instance of that concept if RCC role assertions are present in that ABox.

In principle, the specific properties of qualitative spatial relationships resp. roles cannot be captured completely within $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (we will elaborate 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 "real" subsumption relationships, and that the TBox might be incoherent without being noticed. We believe that careful modeling can avoid this. Another option is to compute the *taxonomy* of the TBox "offline" with a specialized (non-optimized, prototypically) DL system with space semantics such as an $\mathcal{ALCI}_{\mathcal{RCC}8}$ prover (see [Wes03b]), even though this DL is undecidable, [LW04].[3] The deduced implied subsumption relationships can be made syntactically explicit by means of additional TBox axioms, and the augmented TBox can replace the original one.

Much more importantly in our setting is the observation that *ontology-based query answering* can still be achieved in a way that correctly reflects the semantics of the spatial relationships with RacerPro. Consider the *instance retrieval query* $(?x, public\_park\_containing\_a\_lake)$ on the ABox

[3] One could at least try to compute the taxonomy.

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

If this ABox has been computed from the concrete geometry of the map, then it must also contains $(i,k) : NTPPI$, since each RCC network which has been computed from a spatial constellation which shows $(i,j) : TPPI$ and $(j,k) : NTPPI$ must also show $(i,k) : NTPPI$. In order to retrieve the instances of *public_park_containing_a_lake*, we check each individual separately. Let us consider i. Checking that i is an instance of *public_park_containing_a_lake* is reduced to checking the unsatisfiability of the ABox

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

This ABox is obviously unsatisfiable and thus, i is a *public_park_containing_a_lake*.

Regarding concepts that "contain or imply" *universal role or number restrictions,* 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)* in order to keep the semantics of the RCC roles.

To close the ABox $\mathcal{A}$ w.r.t. the RCC role assertion, we count for each individual $i \in \mathsf{individuals}(\mathcal{A})$ and each RCC role R the number of R-successors, $n = |\{\, j \mid\ (i,j) : R \in \mathcal{A}\,\}|$, and add so-called *number restrictions* $i : (\leq\ R\ n) \sqcap (\geq\ R\ n)$ to $\mathcal{A}$. This assertion is satisfied in an interpretation $\mathcal{I}$ iff $n = \{\, j \mid\ (i^{\mathcal{I}}, j) \in R^{\mathcal{I}}\,\}$; thus, i must have exactly n R successors in every model.

In combination with the UNA, this enables a *closed domain reasoning* on the individuals which are mentioned in the RCC role assertions and thus prevents the reasoner from generating "new anonymous RCC role successors" in order to satisfy an existential restriction such as $\exists NTPPI.lake$. In order to satisfy $\exists NTPPI.lake$, it must then necessarily *reuse* one of the existing individuals in the ABox, thus the domain is "closed" [Wes03a]. Let us demonstrate this using the concept

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

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

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

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

Thus, we have argued that spatio-thematic ontology-based query answering can be done on such an ABox representation of the DISK.

Moreover, we must not resort to simple instance retrieval queries, but can use a powerful ABox QL such as nRQL. With nRQL we can pose queries (here in mathematical syntax) such as

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

Note that $\forall adjacent.\neg industrial\_area$ is a complex query concept; we could have put

a concept definition such as $healthy\_living\_area \doteq \forall adjacent.\neg industrial\_area$ into the TBox and used the atom $healty\_living\_area(?living\_area)$ instead. The concrete nRQL syntax will be shown and used below.

However, the discussed ABox representation has the *following drawbacks:*

• **1.** The size of the generated ABoxes is huge. Since the RCC network is explictly encoded in the ABox, the number of required role assertions is quadratic in the number of map objects (resp. individuals).

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

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

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

• **5.** Most importantly, we have demonstrated that this kind of ontology-based query answering only works if the domain is closed. However, DL systems and thus RacerPro are not really good at closed domain reasoning, since the *open domain assumption* is made in DLs. In contrast, since the geometry of the map is completely specified, there is neither unknown nor underspecified spatial information. This motivates the classification of the map as "spatial data", in contrast to "spatial information". Thus, regarding the spatial aspects, deduction or logical inference is not needed. Instead, *model checking* would be sufficient regarding the spatial aspects. All these points thus motivate

• **Representation Option 2 – Use a Map Substrate:** Due to the problems with spatio-thematic concepts and since closed domain reasoning is anyway all that we can achieve here, it seems more appropriate to represent the spatial aspects *primarily* in a different representation layer which we can *associate* with an ABox, a kind of "spatial database", that contains instances of spatial datatypes (polygons etc.). We already mentioned that the geometry of the map must be represented in the extensional component anyway (at least for presentation purposes). This also reflects appropriately that there is neither unknown nor underspecified information regarding the spatial aspects of the map. Everything is explictly given (the "logical theory" of the map is thus complete). This is a reasonable assumption in this scenario.

In the following, this spatial medium is called an *SBox.* If we say that 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. 3(b), we call it a *map substrate.* It is shown that some ABox individuals have corresponding instances in the SBox, and vice versa, since the mapping function is partial and injective. This association / mapping function is called the $*$-function in the following.

If spatial information about the map is now primarily kept in the SBox, then it is no longer available for ABox reasoning. Thus, nRQL (or instance retrieval) queries are no

longer sufficient to address the spatial aspects – we will thus extend nRQL to become a hybrid spatio-thematic QL, offering also *spatial query atoms* to query the SBox: SnRQL. The SnRQL query answering engine will combine the retrieved results from the SBox and ABox part of the hybrid map substrate representation. The purely thematic part of the query will be a plain nRQL query which will be answered on the ABox, and the spatial part of the query will contain spatial atoms which are evaluated on the SBox.

Since the SBox represents the geometry of the map, it can evaluate the requested spatial aspects on the SBox "on the fly" during query evaluation by means of *inspection methods* (see Page 5). The SBox provides a *spatial index*, supporting the efficient computation of spatial relationships by means of spatial selection operations. Computed spatial aspects can also be made explicit and "materialized" in order to avoid repeated re-computation (e.g., computed RCC relations can be materialized as edges).

By means of query rewriting and expansion, the hybridness of SnRQL can be made transparent for the users. Ideally, a user can abstract from the details of the DISK representation in the extensional component of the DLMAPS system. He/she must not known how and where a special aspect of the DISK is physically represented (in the ABox or SBox) in order to be able to formulate queries which return the intended results. The exploited query expansion and rewriting procedures are currently hard-coded, though.

In order to provide these flexible representation options, to "break out" of the strict DL ABox framework, but nevertheless keep a clear mathematical semantics, we base the DLMAPS IS on a semi-structured graph-based *substrate data model* which provides this flexibility. As explained, it serves both as a mediator and software abstraction ("semantic middle ware"), but also enables us to formally describe and combine different representation layers to build hybrid representations such as a map substrate. Since ABoxes play a role here as well, the data model must also generalize the notion of an ABox:

Definition 1 A *substrate* is an edge- and node-labeled directed graph (V, E, L_V, L_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 vertices 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. ■

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* (in appropriate syntax). For example, we can consider an ABox as a substrate if we identify V with the ABox individuals, E with the pairs of individuals mentioned as arguments in role assertions in that ABox, $\mathcal{L}_\mathcal{V}$ with the set of $\mathcal{ALC}$ concepts, and $\mathcal{L}_\mathcal{E}$ with the set of $\mathcal{ALC}$ role names closed under conjunction. Using the first-order perspective, V is a set of constant symbols, and L_V and L_E are *indexing functions*. Obviously, an encoding of $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{V}$ into first-order logic is possible. Moreover, an associated TBox manifests itself in additional first-order sentences which are assumed to be "intrinsically encoded" into the substrate as well.

We do not claim that this data model is interesting from a theoretical perspective; its "generic definition" is of course also its weakness. Thus, it must be specifically *instantiated*. However, the given definition enables a formal specification of the semantics of our substrate representation and query answering framework on which we have based the DLMAPS system.

The data model is somehow inspired by the work on $\mathcal{E}$-Connections [KLWZ04] or the tableaux data structure [HST99], as well as by RDF. However, we feel that we need more flexibility than, e.g., $\mathcal{E}$-Connections or RDF can provide; e.g., RDF does not allow us to describe the geometry of map object. Moreover, a substrate is not simply a "graph database", since DL ABoxes are considered part of the framework and thus, inference is required in order to answer queries. We have the feeling that first-order logic is all we need here in order to formally describe the data model, it semantics and its inference services as well as the *generic substrate query language* (see below).

As just explained, an ABox can be seen as a specialized substrate. The substrate establishes a graph perspective on the ABox (by means of the indexing functions). Moreover, an SBox can be seen as a specialized substrate as well. Here, the nodes are instances of spatial datatypes. We need not leave the framework of first-order logic if we agree that the geometry of such nodes can be described using an appropriate *geometry description language*. We do not want to go into the details (of logical encoding of instances of spatial data types) here.

In order to make use of the substrate representation, a *substrate QL* is needed. The *substrate QL framework* enables the creation of specialized substrate QLs which can be tailored for special kinds of substrates (e.g., ABox, SBoxes). This QL framework is based on the general notion of *ground query atom entailment*. All that matters is that a *notion of logical entailment* ($\models$) between a substrate S and a *ground query atom for* S is defined and decidable. Thus, give the (unary and binary) ground query atoms $P(i)$ and $Q(i,j)$ for $i,j \in V$, $S \models P(i)$ and $S \models Q(i,j)$ must be defined and decidable.

The $\models$ relation is, in principle, the standard first-order one. The $\models$ relation defined for a substrate can *intrinsically encode a rich background theory,* for example, the additional axioms of a background TBox, or *domain closure axioms,* or even the full set of *Clark completion axioms.* These additional axioms are thus not explicitly represented in the substrate as sentences, but are assumed to be part of the inherent structure of the substrate. The models relation is thus further constrained. For example, in the case of an SBox, deciding the $\models$ relation boils down to simple *model checking*. We will give an example for such a background theory when we discuss the RCC substrate.

The substrate QL framework provides a great flexibility, extensibility and adaptability, since specialized query atoms can be tailored for specific instantiations of the substrate data model. If only a notion of entailment is defined and decidable for these specialized atoms, then the substrate query answering engine immediately supports the evaluation of these atoms. In fact, it suffices to overload a single CLOS multimethod `entails-p`. Moreover, decidability of the QL is automatically guaranteed by the framework. *Complex substrate queries* are combined from query atoms by means of body constructors. The substrate QL framework provides the following constructors: negation as failure (NAF, "\"), conjunction (";" and "$\wedge$"), union ("$\vee$"), as well as a *projection operator* ("π"). The semantics (and decidability) of these generic QL operators will become clear if we consider nRQL, which is a specialized substrate QL tailored for RacerPro ABoxes; it thus shares the same semantics for the body constructors as all substrate QLs.

Definition 2 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

$* : V_1 \mapsto V_2$. ■

The DLMAPS system is designed in such a way to work on ABoxes, SBoxes, or *map substrates:*

Definition 3 A *map substrate* is a hybrid substrate $(S_1, S_2, *)$, where S_1 is an SBox, and S_2 is an ABox (substrate). ■

Given the hybrid representation, 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 easily, we simply prefix variables in query atoms which for S_2 with a "?$*$" instead of "?"; the same applies to constants / individuals. Intuitively, the *bindings* which will be established for variables must reflect the $*$-function: If $?x$ is bound to $i \in V_1$, then $? * x$ will automatically be bound to $*(i) \in V_2$, and vice versa (w.r.t. $*^{-1}$). Such a binding is called $*$-*consistent*. We will only consider $*$-consistent bindings. The notion of $*$-consistent bindings is also depicted in Fig. 3(b).

Assume we use a map substrate for the DISK representation now. Then, the nRQL example query given above will become a hybrid SnRQL query; nRQL atoms now use $*$-prefixed variables (since the ABox is S_2, and the SBox is S_1):

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

Please note that the last conjunct in the query is the "equivalent" of the nRQL atom $(\forall adjacent.\neg industrial\_area)(?living\_area)$.

By means of the definition of the semantics of the QL (see below) it will become clear that $\pi(? * living\_area)\ (adjacent(? * living\_area, ? * industrial\_area), \ldots)$ is in fact equivalent to (resp. has the same extension as) $ans(? * living\_area) \leftarrow adjacent(? * living\_area, ? * industrial\_area), \ldots$. This subquery therefore retrieves all map objects which are adjacent to an industrial area. The NAF operator *complements* this subquery result (see next Section on nRQL semantics). Thus, the binding to $?living\_area$ must be an instance of *living_area* which *does not have a (known) adjacent industrial area.* This implies that that all known adjacent areas are not industrial areas.

• **Representation Option 3 – Use an ABox + RCC Substrate:** We have argued that in general we cannot completely capture the inherent properties of the RCC relations in an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ ABox, since $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ lacks the required expressivity. However, we have argued that (see Option 1) at least for ontology-based spatio-thematic query answering a $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ ABox can still be used, given that the RCC network in the ABox was computed from a concrete map. The RCC network is thus complete (lacks no edges) and automatically RCC consistent. Moreover, we must enable closed domain reasoning for the RCC roles.

In order to make a comparable query answering functionality available to other users of the RacerPro system without having to add the whole SBox functionality to RacerPro (spatial datatypes and spatial indexes, etc.), we devise yet another kind of substrate, the *RCC substrate*, which captures the semantics of the RCC relations by exploiting

techniques from *qualitative spatial reasoning.* Users of RacerPro can associate an ABox $\mathcal{A}$ with an RCC substrate $\mathcal{RCC}$ by means of a hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ and query this hybrid substrate with *nRQL + RCC query atoms* (see below). Note that, unlike a map substrate, there the ABox is S_1 and thus the "primary" substrate.

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

Definition 4 An RCC substrate $\mathcal{RCC}$ is a substrate such that V is a set of RCC nodes with $\mathcal{L}_\mathcal{V} = \emptyset$, and $\mathcal{L}_\mathcal{E} = 2^{\{EQ,DC,EC,PO,TPP,TPPI,NTPP,NTPPI\}}$. ■

An edge label represents a *disjunction of RCC base relations*, representing coarser or even unknown knowledge regarding the spatial relation. 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 6) 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 possible values for T. In general, T will not be a base relation, but a set: $\{T_1, \ldots T_n\}$. The RCC table is given as a set $\mathcal{RCC}_\mathcal{T}$ of sentences of the form $\{R \circ S = \{T_1, \ldots, T_n\}, \ldots\}$.

An RCC network $\mathcal{RCC}$ (viewed as set of first-order ground atoms) containing only base relations is said to be consistent iff it satisfies a number of first-order sentences:

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

For example, the network $\mathcal{RCC} = \{NTPP(a,b), DC(b,c), PO(a,c)\}$ is inconsistent, because if a is contained in b (atom $NTPPI(a,b)$), and b is disconnected from c (atom $DC(b,c)$), then a must be disconnected from c as well. The *RCC8 composition table* contains the axiom $NTPP \circ DC = DC$. Thus, $\mathcal{RCC}' \models DC(a,c)$, which contradicts $PO(a,c)$, due to the JEPD property. *Entailment of RCC relations* or RCC ground query atoms can be *reduced to inconsistency checking:* $\mathcal{RCC}' \models R(a,b)$ **iff** $\mathcal{S} \cup (\{EQ, DC, EC, PO, TPP, TPPI, NTPP, NTPPI\} \setminus R)$ **is not consistent.** An RCC network is consistent iff at least one of its *configurations* is consistent. A configuration of an RCC network is obtained by choosing (and adding) one disjunct from every non-base relation in that network. Thus, a configuration contains only base relations. Consider now $\mathcal{RCC} = \{NTPP(a,b), DC(b,c)\}$. We have $\mathcal{RCC}' \models DC(a,c)$, since $\mathcal{RCC}' \cup \{(EQ \vee EC \vee PO \vee TPP \vee TPPI \vee NTPP \vee NTPPI)(a,c)\}$ is inconsistent, because its *configurations* $\mathcal{RCC}' \cup \{EQ(a,c)\} \ldots \mathcal{RCC}' \cup \{NTPPI(a,c)\}$ are all inconsistent.

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 $(\mathcal{A}, \mathcal{RCC}, *)$

$$\mathcal{A} = \{hamburg : german\_city, paris : french\_city, france : country, germany : country\}$$

with

$$\mathcal{RCC} = \{NTPP(*hamburg, *germany), EC(*germany, *france), NTPP(*paris, *france)\}$$

and with the obvious (trivial) mapping $*$

$$* = \{(hamburg, *hamburg), (paris, *paris), (france, *france), (germany, *germany)\}.$$

Then, the query

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

will correctly return

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

4 The nRQL ABox Query Language

The nRQL ABox QL is now described in some detail here, since this is the QL which is used and extended to a spatial QL in this paper.

At a first glance, it will seem that nRQL is in a line of QLs which is closely related to Horn logic (query) languages such as Datalog: After all, nRQL provides some form of conjunctive queries. As such, one might criticize nRQL for not being "state of the art in logic programming techniques". However, considering nRQL as some-kind of non-recursive Datalog with Negation is inappropriate and *pragmatically misleading,* since the language has been designed under a different perspective. nRQL is an *instantiation* of a more general QL framework, the *substrate QL framework*. We claim that this framework provides more flexibility and options for extensions than, for example, Datalog, and demonstrate this in this paper. In order to support this claim, it is thus crucial that we formally define syntax and semantics of nRQL resp. of the generic substrate QL framework.

In the following we will use the *concrete syntax of nRQL.* A nRQL query consists of a *query head* and a *query body*:

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

has the head `(?x ?y)` and the body `(and (?x woman) (?x ?y has-child))`. The query returns all mother-child pairs from the ABox which is queried. In a nutshell, nRQL can be characterized as follows:

• **Variables and individuals** can be used in queries. The variables range over the individuals of an ABox (this is called *active domain semantics*), and are bound to those ABox individuals which *satisfy* the query. The notion of satisfiability of a query used in nRQL is defined in terms of logical entailment. A variable is bound to an ABox individual iff it can be proven that this binding holds in *all* models of the knowledge base. Returning to our example query body `(and (?x woman) (?x ?y has-child))`, `?x` is only bound to those individuals which are instances of the concept `woman` having a *known* child `?y` in *all* models of the KB.

nRQL distinguishes variables which must be bound to *differently named individuals* (prefix `?`, e.g., `?x, ?y` cannot be bound to the same ABox individual) from variables for which this does not hold (prefix `$?`, e.g., `$?x, $?y`). Individuals from an ABox are identified by using them directly in the query, e.g. `betty`.

• **Different types of query atoms** are available: these include concept query atoms, role query atoms, constraint query atoms, and same-as query atoms. To give some examples, the atom `(?x (and woman (some has-child female)))` is a concept

query atom, `(?x ?y has-child)` is a role query atom, `(?x ?x (constraint (has-father age) (has-mother age) =))` is a constraint query atom (asking for the persons `?x` whose parents have equal age), and `(same-as ?x betty)` is a same-as query atom, enforcing the binding of `?x` to `betty`.

As the given example concept query atom demonstrates, it is possible to use complex concept expressions within concept query atoms. Regarding role query atoms, the set of role expressions is more limited. However, it is possible to use inverted roles (e.g., role expressions such as `(inv R)`) as well as *negated roles* within role query atoms. Note that negated roles are not supported in the concept expression language of $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$; thus, they are only available in nRQL. For example, the atom `(?x ?y (not has-father))` will return those bindings for `?x, ?y` for which RacerPro *can prove* that the individual bound to `?x` cannot have the individual bound to `?y` as a father. If the role `has-father` was defined as having the concept `male` as a range, then at least all pairs of individuals in which `?y` is bound to a `female` person are returned, given `male` and `female` can be proven to be disjoint.

• **Complex queries** are built from query atoms using the boolean constructors `and, union, neg, project-to`. This holds for *all substrate QLs*. We have already seen an example: `(and (?x woman) (?x ?y has-child))` is a simple *conjunctive query body.* These constructors can be combined in an arbitrary (orthogonal) way to create complex queries. This is why we call nRQL an orthogonal language.

The `neg` operator implements a *negation as failure semantics (NAF)*: `(neg (?x woman))` returns all ABox individuals for which RacerPro *cannot prove* that they are instances of `woman`. Thus, `(neg (?x woman))` returns the complement set of `(?x woman)` w.r.t. the set of all ABox individuals. If used in front of a role query atom, e.g. `(neg (?x ?y has-child))`, then this returns the complement of `(?x ?y has-child)` (w.r.t. to all pairs of ABox individuals), i.e. all pairs of individuals which are not in the extension of `has-child` in all models. The semantics of nRQL ensures that DeMorgan's Laws hold: `(neg (and` $a_1 \ldots\ a_n$`))` is equivalent to `(union (neg` a_1`) ... (neg` a_n`))`.

Note that `(?x (not woman))` has a different semantics from `(neg (?x woman))`, since the former returns the individuals for which RacerPro *can prove* that they are *not instances* of woman, whereas the latter returns all instances for which RacerPro *cannot prove* that they are *instances* of woman.

• **Support for retrieving *told values* from the CD:** Suppose that `age` is a so-called *concrete domain attribute* of type integer. Thus, the `age` attribute fillers of a certain individual must be concrete domain values of type `integer`. We can use the following query to retrieve all adults as well as their ages: `(retrieve (?x (told-value (age ?x)) (?x (min age 18))))`, and a possible answer might be `(((?x michael) ((told-value (age michael)) 34)))`. Please refer to [WM05, KG06] for more details and the design rationale.

• **Support for CD constraint checking :** The so-called *constraint query atoms* allow one to "compare" concrete domain attribute fillers of different individuals. Consider the query

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

which returns the list of women and their ages. The women are required to have children whose fathers are at least 8 years older than their mothers. Note that `(has-father age)` denotes a "path expression": starting from the individual bound to `?y` 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 evaluate the compound concrete domain predicate `(<= (+ age-2 8) age-1)`. Here, `age-2` refers to `(has-mother age)`, and `age-1` refers to `(has-father age)`. Note that the suffixes `-1, -2` have been added to the `age` attribute in order to differentiate the two values. Obviously, this mechanism is not needed if the two chains are ended by different attributes.

• **Special support for querying OWL documents**, e.g., retrieving told datatype value fillers of OWL datatype and OWL annotation properties. Retrieval of these datatype values is supported in a similar style as in the concrete domain case, by means of concept query atoms and head projection operators. We do not go into detail here, please refer to [KG06].

• **The body projection operator (`project-to`):** Sometimes this operator is required in order to reduce the "dimensionality" of a tuple set, for example, before computing its complement with `neg`.

Let us motivate the necessity for such an operator: Consider `(retrieve (?x) (and (?x mother) (?x ?y has-child)))`. This query returns all mothers having a *known child* in the ABox. Now, how can we query for mothers which do *not* have a *known* child? Our first attempt will be the query `(retrieve (?x) (and (?x mother) (neg (?x ?y has-child))))`. A bit of thought and recalling that `(neg (?x ?y has-child))` returns the complement set of `(?x ?y has-child)` w.r.t. the Cartesian product of all ABox individuals will reveal that this query doesn't solve the task. In a second attempt, we will probably try `(retrieve (?x) (neg (and (?x mother) (?x ?y has-child))))`. However, due to DeMorgan's Law and nRQL's semantics, this query is equivalent to `(retrieve (?x) (union (and (neg (?x mother)) (?y top)) (neg (?x ?y has-child))))` – first the union of two two-dimensional tuple sets is constructed, and then only the projection to the first element of these pairs (`?x`) is returned. Obviously, this set contains also the instances which are *not* known to be mothers, which is wrong as well. Thus, the need for the projection operator becomes apparent: `(retrieve (?x) (and (?x mother) (neg (project-to (?x) (?x ?y has-child)))))` solves the task. This body projection operator was not present in earlier versions of nRQL, special syntax was introduced to address these problems, namely the special unary atoms `(?x (has-known-successor has-child))`, `(?x NIL has-child)` and `(NIL ?X child-of)`. These atoms (which still work) can now be seen as "syntactic sugar" for the bodies `(project-to (?x) (?x ?y has-child))`, `(neg (project-to (?x) (?x ?y has-child)))` and `(neg (project-to (?x) (?y ?x has-child)))`. The `project-to` operator can be used at any position in a query body.

We can now define syntax and semantics of nRQL. Please note that all substrate QLs share in principle the same syntax and semantics, only *atom* (and possibly also *head_projection_operator*) are redefined appropriately, as already mentioned:

Definition 5 (Syntax of nRQL) A nRQL query has a *head* and a *body*.

A *head* can contain the following (note that {a|b} represents `a` or `b`):

head	:=	(*head_entry**)
object	:=	*variable* | *individual*
variable	:=	a symbol beginning with "?"
individual	:=	a symbol
head_entry	:=	*object* | *head_projection_operator*
head_projection_operator	:=	(*cd_attribute object*) |
		`(told-value (`*cd_attribute object*`))` |
		`(told-value (`*datatype_property object*`))` |
		`(annotations (`*annotation_property object*`))`

The *body* is defined as follows:

body	:=	*atom* | `(` {`and` | `union`} *body**`)` | `(neg` *body*`)` |
		`(project-to (`*object**`)` *body*`)`
atom	:=	(*object concept_expr*) | (*object object role_expr*) |
		(*object object* `(constraint` *chain chain constraint_expr*`))` |
		`(same-as` *object object*`)`
chain	:=	(*role_expr** *cd_attribute*)

The "bridge" to the RacerPro syntax is given by the following rules:

concept_expr	:=	a RacerPro concept, with some extensions for OWL
role_expr	:=	a RacerPro role or the *special role* `equal` |
		`(inv` *role_expr*`)` | `(not` *role_expr*`)`
constraint_expr	:=	a (possibly compound) RacerPro concrete domain predicate, e.g. `(< (+ age-1 20) age-2)`
cd_attribute	:=	a RacerPro concrete domain attribute
datatype_property	:=	a RacerPro role used as OWL datatype property

■

Note that nRQL permits the use of negated roles in role atoms, as well as the special role `equal`. The `equal` role can only be used in nRQL, not in $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ concept expressions. The semantics of `equal` is fixed: $\texttt{equal}^{\mathcal{I}} =_{def} \mathcal{ID}_{2,\Delta^{\mathcal{I}}} = \{ \langle i, i \rangle \mid i \in \Delta^{\mathcal{I}} \}$.

Definition 6 (Semantics of nRQL) Let $\mathcal{A}$ be a RacerPro ABox, and $\mathcal{T}_{\mathcal{A}}$ denote its associated TBox. Denote the set of individuals used in $\mathcal{A}$ with $\mathsf{Inds}_{\mathcal{A}}$.

Let q be a nRQL query *body*. The function $\mathsf{vars}(q)$ is defined inductively:
$\mathsf{vars}(\texttt{(}x \ \textit{concept\_expr}\texttt{)}) =_{def} \{x\}$, $\mathsf{vars}(\texttt{(}x_1 \ x_2 \ \textit{role\_expr}\texttt{)}) =_{def} \{x_1, x_2\}$,
$\mathsf{vars}(\texttt{(}x_1\ x_2\ \texttt{(constraint ...)))} =_{def} \{x_1, x_2\}$,
$\mathsf{vars}(\texttt{(\{ and | union | neg \}}\ q_1 \ldots q_m\texttt{)}) =_{def} \bigcup_{1 \leq i \leq m} \mathsf{vars}(q_i)$, **BUT**
$\mathsf{vars}(\texttt{(project-to (}x_1 \ldots x_m\texttt{) ...)}) =_{def} \{x_1 \ldots x_m\}$. Thus, vars "stops at projections".

Assume that $\langle x_{1,q}, \ldots, x_{n,q} \rangle$ is a lexicographic enumeration of $\mathsf{vars}(q)$. Denote the ith element in this vector with $x_{i,q}$, indicating its position in the vector.

Let $\mathcal{T}$ be a set of n-ary tuples $\langle t_1, \ldots, t_n \rangle$ and $\langle i_1, \ldots, i_m \rangle$ be an *index vector* with $1 \leq i_j \leq n$ for all $1 \leq j \leq m$. Then we denote the set $\mathcal{T}'$ of m-ary tuples

with $\mathcal{T}' =_{def} \{ \langle t_{i_1}, \ldots, t_{i_m} \rangle \mid \langle t_1, \ldots, t_n \rangle \in \mathcal{T} \} = \pi_{\langle i_1,\ldots,i_m \rangle}(\mathcal{T})$, called the *projection* of $\mathcal{T}$ to the components mentioned in the index vector $\langle i_1, \ldots, i_m \rangle$. For example, $\pi_{\langle 1,3 \rangle}\{\langle 1,2,3 \rangle, \langle 2,3,4 \rangle\} = \{\langle 1,3 \rangle, \langle 2,4 \rangle\}$.

Let $\vec{b} = \langle b_1, \ldots, b_n \rangle$ be a *bit vector* of length n, $b_i \in \{0,1\}$. Let $m \leq n$. If $\vec{b}$ is a bit vector which contains exactly m ones, and $\mathcal{B}$ is a set, $\mathcal{T}$ 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

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

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

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

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

In order to simplify the specification of the semantics, the query body q first undergoes some syntactical transformations. In a first step, q is rewritten by consistently replacing all its individuals with representative (fresh) variables throughout the body: If the individual i has been replaced with the variable x_i, then we also add the conjunct `(same-as` x_i i`)` to q, yielding a body of the form `(and` q `(same-as` x_i i`)` $\ldots$`)`. In a second step, the role chains (see syntax rule *chain*) possibly present in constraint query atoms are decomposed. This means they are replaced by conjunctions of role query atoms such that only concrete domain attributes remain in chains of constraint query atoms. Fresh variables are used for this purpose.

We can now define the semantics of the different query atoms, being part of q'. Keep in mind that $\langle x_{1,q'}, \ldots, x_{n,q'} \rangle$ is the variable vector of q' and that n is the total number of variables returned by $\mathsf{vars}(q')$. The semantics for the different nRQL query atoms is given as:

$$\begin{aligned}
(q'_{x_i}\ \mathit{concept_expr})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{concept_instances}(\mathcal{A}, \mathit{concept_expr})) \\
(q'_{x_i}\ q'_{x_j}\ \mathit{rolen_expr})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{role_pairs}(\mathcal{A}, \mathit{role_expr})), \text{ if } i \neq j \\
(q'_{x_i}\ q'_{x_i}\ \mathit{role_expr})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{role_pairs}(\mathcal{A}, \mathit{role_expr}) \cap \mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}}) \\
(\texttt{same-as}\ q'_{x_i}\ \mathit{ind})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\{\mathit{ind}\}) \\
(\texttt{same-as}\ \mathit{ind}\ q'_{x_i})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\{\mathit{ind}\}) \\
(\texttt{same-as}\ q'_{x_i}\ q'_{x_j})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}}), \text{ if } i \neq j \\
(\texttt{same-as}\ q'_{x_i}\ q'_{x_i})^{\mathcal{E}} &=_{def} \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}})
\end{aligned}$$

$$\begin{aligned}
&(q'_{x_i}\ q'_{x_j}\ (\texttt{constraint}\ \mathit{attrib}_1\ \mathit{attrib}_2\ P))^{\mathcal{E}} =_{def} \\
&\qquad \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{predicate_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P)), \text{ if } i \neq j \\
&(q'_{x_i}\ q'_{x_i}\ (\texttt{constraint}\ \mathit{attrib}_1\ \mathit{attrib}_2\ P))^{\mathcal{E}} =_{def} \\
&\qquad \chi_{\mathsf{Inds}_{\mathcal{A}},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{predicate_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P) \cap \mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}})
\end{aligned}$$

This definition uses some auxiliary functions, providing the bridge to the classical ABox retrieval functions offered by RacerPro. These functions have the standard DL semantics in terms of logical entailment:

$$\begin{aligned}
&\mathsf{concept_instances}(\mathcal{A}, \mathit{concept_expr}) =_{def} \\
&\qquad \{\, i \mid\ i \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models \mathit{concept_expr}(i) \,\} \\
&\mathsf{role_pairs}(\mathcal{A}, \mathit{role_expr}) =_{def} \\
&\qquad \{\, \langle i, j\rangle \mid\ i, j \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models \mathit{role_expr}(i, j) \,\} \\
&\mathsf{predicate_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P) =_{def} \\
&\qquad \{\, \langle i, j\rangle \mid\ i, j \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models \\
&\qquad\qquad\qquad \exists x, y : \mathit{attrib}_1(i, x) \wedge \mathit{attrib}_2(j, y) \wedge P(x, y) \,\}
\end{aligned}$$

Moreover, a nRQL *role expression (role_expr)* can also be a *negated* (or inverse) role. In the case of $\mathit{role\_expr} = \texttt{equal}$ we have $(\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models \texttt{equal}(i, j)$ iff $i^{\mathcal{I}} = j^{\mathcal{I}}$ in all models $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $(\mathcal{A}, \mathcal{T}_{\mathcal{A}})$, and in the case of $\mathit{role\_expr} = \neg\texttt{equal}$ we must have $i^{\mathcal{I}} \neq j^{\mathcal{I}}$ in all models. The predicates P are made from the vocabulary offered by RacerPro for building CD predicates (e.g., $<(+(\mathit{age}_2, 8), \mathit{age}_1)$ is a CD predicate with free variables $\mathit{age}_1, \mathit{age}_2$, and $+$ is a functor with the standard semantics).

The semantics of complex nRQL bodies can be defined easily now:

$$\begin{aligned}
\texttt{(and}\ q_1 \ldots q_i\texttt{)}^{\mathcal{E}} &=_{def} \bigcap_{1 \le j \le i} q_j^{\mathcal{E}} \\
\texttt{(union}\ q_1 \ldots q_i\texttt{)}^{\mathcal{E}} &=_{def} \bigcup_{1 \le j \le i} q_j^{\mathcal{E}} \\
\texttt{(neg}\ q\texttt{)}^{\mathcal{E}} &=_{def} (\mathsf{Inds}_{\mathcal{A}})^n \setminus q^{\mathcal{E}} \\
\texttt{(project-to (}x_{i_1,q} \ldots x_{i_k,q}\texttt{)}\ q\texttt{)}^{\mathcal{E}} &=_{def} \pi_{\langle i_1, \ldots, i_k\rangle}(q^{\mathcal{E}})
\end{aligned}$$

So far we have specified the semantics of a query body. To get the answer of a query, the *head* has to be considered. This can be seen as a further projection of $q^{\mathcal{E}}$ to the variables mentioned in the head. If the head contained an individual, then this individual has also been replaced by the representative variable in the head. If a head projection operator is encountered, the functional operator is applied to the binding of the argument variable. The value is included in the query answer (producing nested binding lists); a more formal definition of this function application is omitted here. ■

5 Hybrid Spatio-Thematic Queries

According to the discussion in Section 3, we are either using a single ABox $\mathcal{A}$ (option 1), a (hybrid) map substrate $(\mathit{SBox}, \mathcal{A}, *)$ (option 2), or or a hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ (option 3) for DISK representation in the DLMAPS system. nRQL is used in all settings. We already explained how we make a substrate QL such as nRQL hybrid: A hybrid query uses now two kinds of query atoms, those for S_1, and those for S_2. Variables / individuals in atoms for S_2 are simply prefixed with $*$. Then, in a query body, any combination of atoms for S_1 and S_2 is permitted, and variables are bound in a $*$-consistent way, see Page 11.

Basically, all substrate QLs share the same *body* syntax (Def. 5). The *atom*s are substrate specific, as explained. Also the head projection operators can be specific. The

semantics of complex query bodies in Def. 6 is shared by all substrate QLs. However, each specialized atom must define its own extension by means of the dedicated $\models$ relation; e.g., for an arbitrary *atom*, its extension $atom^{\mathcal{E}}$ must be defined (w.r.t. substrate S). That's all. For a hybrid query language, we additionally require that the computed binding tuples respect the $*$-function, that is, computed bindings must $*$-consistent. We do not go into formal details here.

Which spatio-thematic QL is now applicable for the different representation options in the DLMAPS system?

• **For option 1:** We can only use plain nRQL, as explained.

• **For option 2:** The resulting hybrid QL is called SnRQL. It provides the following spatial atoms in addition to nRQL:

• **RCC atoms:** Atoms such as `(?x ?y (:tppi :ntppi))` etc. *Spatial prepositions* such as `:contains, :adjacent, :crosses, :overlaps, :flows-in` are available.

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

• **Epsilon Atoms:** `(?x ?y (:inside-epsilon <min> <max>))`. With that atom, all objects `?y` are retrieved, such that `?y` is contained within the *buffer zone* of size $[min; max]$ around `?x`. This buffer zone consists of all points (x, y) whose *shortest distance to the fringe of* `?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 polygons whose area is in $[100; 1000]$. Also `:length` is understood.

Moreover, simple type checking atoms such as `(?x :is-polygon)`, `(?x :is-line)` etc. are available.

To give a final example, consider this SnRQL query in concrete syntax 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)))))
```

The extensions of the atoms are computed on the fly from the geometry with *spatial inspections methods* (see Page 5). As argued, this can be understood as a kind of (spatial) model checking.

• **For option 3,** the resulting hybrid QL is called *nRQL + RCC atoms*. This language can only offer RCC atoms, since the geometry of the map is not represented. The syntax of the SnRQL RCC atoms is identical to the RCC atoms just discussed.

6 Conclusion

Building ontology-based IS 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. Since decidability is not always easy to achieve it is thus of even more importance to identify pragmatic and practical solutions which, even though if they do not exploit or advance the latest theoretical state-of-the-art techniques in DL research, can nevertheless be considered an advance w.r.t. the current state-of-the-art of 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 can be defined provides a great deal of flexibility for building similar ontology-based IS. Moreover, the implemented functionality is available for other users and system designers. Thus, we believe that these pragmatic solutions can be called empirically successful. DLMAPS can be found under `http://www.sts.tu-harburg.de/~mi.wessel/dlmaps/dlmaps.html`.

References

[HLM99] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[HM01a] V. Haarslev and R. Möller. RACER System Description. In *Int. Joint Conference on Automated Reasoning*, 2001.

[HM01b] V. Haarslev and R. Möller. The Description Logic ALCNHR+ Extended with Concrete Domains: A Practically Motivated Approach. In *International Joint Conference on Automated Reasoning*, 2001.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proc. International Conference on Logic for Programming and Automated Reasoning*, 1999.

[KG06] Racer Systems GmbH & Co. KG. RacerPro User's Guide 1.9.0. Technical report, Racer Systems GmbH & Co. KG, `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf`, 2006.

[KLWZ04] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-Connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.

[LM05] C. Lutz and M. Milicic. A Tableau Algorithm for DLs with Concrete Domains and GCIs. In *Proc. of the International Workshop on Description Logics*, 2005.

[LW04] C. Lutz and F. Wolter. Modal logics of topological relations. In *Proc. of Advances in Modal Logics*, 2004.

[Wes03a] 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*, 2003.

[Wes03b] M. Wessel. Qualitative Spatial Reasoning with the $\mathcal{ALCI}_{\mathcal{RCC}}$-family – First Results and Unanswered Questions. Technical report, May 2003. Available at `http://www.sts.tu-harburg.de/~mi.wessel/papers/report7.pdf`.

[WM05] M. Wessel and R. Möller. A High Performance Semantic Web Query Answering Engine. In *Proc. International Workshop on Description Logics*, 2005.