Master Thesis

Implementation of a Perfect Rewriting Algorithm for Ontology Based Query Answering over Spatial Databases

Of

Aleksandar Gudov

October 2012

First Examiner	Prof. Dr. rer. nat. habil. Ralf Moeller Institute for Software Systems Hamburg University of Technology
Second Examiner	Prof. Dr. rer. nat. Volker Turau Institute of Telematics Hamburg University of Technology

Declaration of Authorship

I declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research. Each significant contribution to it and quotation from the work of other people has been attributed and referenced. This thesis has not been previously submitted in whole or in part for the award of any degree.

Date: 15.10.2012 Signature:

Contents

1 Introduction

1.1 Motivation

One of the key problems in information systems and particularly in current geographic information systems (GIS) is the effective management of information, since on the one hand the amount of information has increased tremendously and on the other hand, the information becomes more complex. However, the information should be accessible in an efficient, flexible and automated way. In order to fulfill these requirements, the traditional data management systems unfortunately surrender.

Ontology Based Data Access (OBDA) is considered as a suitable, flexible and powerful approach, being able to handle sophisticated data management tasks, by providing access to data, potentially stored in heterogeneous data sources, with the help of a semantic layer in the form of an ontology. In terms of GIS, used for instance in the areas of city planning, transportation networks, urban management, etc., ontologies are appropriate for the formalization of relevant concepts and relations among spatial regions in a conceptual data model, thus providing also the possibility to answer queries. In a nutshell, ontologies give the beneficial flexibility for users to describe their own models on GIS data and formulate easily queries over the data.

However, query answering at the conceptual tier of a geographical or spatial model requires deduction techniques and features, which are difficult to realize due to the size of the persistent data stored in geographic databases. An advocated solution to the problem of query answering over geo-thematic ontologies is to use specific lightweight description logics and query languages in order to keep low complexity of reasoning and provide sufficient expressivity for modeling and querying important features of GIS data.

1.2 Objectives

The aim of this Master Thesis is to evaluate approaches for ontology based query answering over spatial databases. Another objective is to address current issues in ontology based data access and propose solutions for solving or minimizing these problems, by analyzing possible techniques and scientific approaches.

Furthermore, the modeling and reasoning capabilities of DL-Lite(RCC8) with respect to expressivity and efficiency of reason, as well as the querying features of GCQ^{+}, are to be examined in detail. As a next step, the Adapted Perfect Rewriting Algorithm should be

analyzed and consequently implemented within a software system, being able to perform flexible and powerful ontology based querying answering over spatial databases.

1.3 Achievements

The main theoretical contribution of this Master Thesis is that a flexible, efficient, fast and reliable method for performing ontology based query answering over spatial databases can be realized by incorporating the following approaches and techniques:

- The modified logic DL-Lite(RCC8), realized by a weak coupling of Lightweight DLs with the expressive Region Connection Calculus RCC8, can be used to effectively and easily model the conceptual level of the ontology.

- The query language GCQ^+ can be used in order to define complex queries, since it provides support for qualitative spatial query answering.

- A top-down approach for query answering over DL-Lite(RCC8) ontologies with mappings of the form $O=<T, M, D>$, where T is a TBox (a set of axioms), M is a set of mappings and D is a spatial database, can be used by applying an optimized Adapted Perfect Rewriting Algorithm for conjunctive queries, followed by an Unfolding process, where with the help of the mappings M the output query of the reformulation step, being a union of conjunctive queries, is consequently transformed into an SQL query, thus avoiding the materialization of the virtual ABox (a set of assertions) of the form $A(M, D)$. In a nutshell, the initial query can be compiled into an SQL query that is consequently posed to the spatial database, making use of the query optimization techniques provided by current database management systems.

Another practical achievement of this Master Thesis, verifying the latter described theoretical contribution, is the development of an ontology based Query Answering System in Java that provides:

- a framework for representing standard DL-Lite TBox;

- a framework for representing DL-Lite(RCC8) TBox, containing in addition concepts of the form $\exists U_1,U_2.r$, where $r \in Rel_{RCC8}$ and $U \rightarrow loc \mid R \circ loc$;

- a framework for representing conjunctive queries, containing query atoms of concepts, roles or GCQ^+ atoms of the form $\exists U_1,U_2.r(x)$;

- a framework for representing object-to-data mappings, containing mapping assertions of the form $M_{left} \sim> M_{right}$, such that the left part is and SQL query and the right part is a conjunction of atoms over the TBox;

- a Parser for reading user inputs
- a Reasoner for implementing the Original Perfect Rewriting Algorithm and the Adapted Perfect Rewriting Algorithm;
- a Reasoner for implementing the Query Reformulation process and evaluating the output of the Query Reformulation process over a PostgreSQL database.

Furthermore, a performance optimization of the Adapted Perfect Rewriting algorithm has been achieved. The first rewriting rule (cf. Chapter 3.6) of the algorithm has been extended by finding the maximum pairs r_1,r_2 such that $r_1;r_2 \subseteq r_3$ and realizing a reformulation process only w.r.t. these pairs, thus avoiding redundancy and decreasing complexity (cf. Figure 5.10, line 14).

Finally, experiments for testing the Perfect Rewriting Module of the application, using both pure DL-Lite and DL-Lite(RCC8) TBox-es, have been realized. Last but not least, the complete Query Answering Reasoner, including the Reformulation, Unfolding and Evaluation Modules has been evaluated over pure DL-Lite ontologies.

1.4 Chapter Structure

Chapter 2 introduces current issues in information systems. A detailed explanation of the Description Logics, referred as the formal foundations for ontologies, is consequently revealed. The Ontology Based Data Access Control is explained and its core issues are defined. Last but not least, the chapter finishes with a brief comparison and evaluation of current spatial databases and existing OBDA software applications. Chapter 3 presents a problem and requirement analysis. It discusses the trade-off between expressive power and computational complexity of reasoning, as well as different approaches how to link data to ontologies. Then the core ideas of DL-Lite and several DL-Lite extensions are presented. Consequently, the main part of this chapter concentrates on the DL-Lite Combined Geo-thematic Logics, GCQ^+ and the Adapted Perfect Rewriting Algorithm. Finally, the chapter ends with a list of all program and technical requirements, based on the previously analyzed concepts, technologies, solution proposals, and limitations. Chapter 4 and 5 are the main chapters of this Master Thesis, where the practical achievements are explained by presenting the actual design and architecture of the developed GIS application with DL-Lite(RCC8) ontologies. Moreover, Chapter 5 reveals how the core components of the system are realized by describing the most important software components, techniques and issues. Furthermore, the processes query reformulation and query unfolding are explained, followed by overviews of the application outputs and concluding discussions of the achieved results. Chapter 6 continues with a brief discussion how the developed system has been tested and recommends several future research topics and program improvements. This Master Thesis ends with a conclusion chapter that summarizes the main goals and achieved results.

2 Fundamentals and Background

This chapter starts with a short overview of the current issues in information systems. Consequently, the essence of knowledge representation is discussed and a number of knowledge representation techniques and approaches are presented. A detailed explanation of the Description Logics, referred as the formal foundations for ontologies, is consequently revealed in order to provide background information for the ideas, used in the following chapters. The Ontology Based Data Access Control is explained and its core issues are defined. Last but not least, the chapter finishes with a brief comparison that reveals and evaluates current spatial databases and existing OBDA software applications.

2.1 Knowledge Representation Techniques

Traditionally, the complex manipulation of data information has been one of the main topics of interest in the area of knowledge representation in Artificial Intelligence. The core aim of knowledge representation is to formally model knowledge by providing high-level description of the world and thus enabling the process of effectively drawing logical conclusions about the modeled world. The process of modeling is usually realized in complex knowledge representation and reasoning systems, which are based on various formal knowledge representation languages and notions. These systems provide the possibility to implicitly find consequences, based on the explicitly represented knowledge.

The knowledge representation approaches and techniques going back to the 1970's can be classified into two main groups: non-logic-based representations and logic-based formalism. In the second approach, the language, used for knowledge representation, is commonly a variation of a first-order predicate calculus, while the non-logic-based approach often relies on the use of networks, graphical interfaces or various ad hoc data structures. Examples of the non-logic-based technique are Semantic Networks and Frames, which can be also specified as Network-Based Structures [1]. Representatives of the second approach are among others Description Logics (DL), the standard Web Ontology Language (OWL), Datalog and rule-based languages, Predicate Logic, etc. [2]. The main advantage of the logic-based approach is that it is more powerful, more general-purpose and more expressive than the non-logic-based representation technique.

The core building parts of the network-based representation structures are nodes and links, which correspondingly depict concepts and roles. A concept is a class or set of particular objects and a role represents the connection or relation among these objects. If the relation, i.e. role is more complex, then it can be also represented as a node, but with a different shape. Moreover, concepts can have further characteristics such as attributes or properties, which are attached to the corresponding nodes in the network. An example network is illustrated in

Figure 2.1, where knowledge about cats, pets, animals, mammals and carnivores can be depicted. This network is also referred as a terminology, depicting the described world that consists of concepts, roles and attributes. The roles are represented by the blue arrows. For instance, the link between the concept *cat* and *carnivore* means that *a cat is a carnivore*. This type of connection is also called an "IS-A" relationship [1]. It is important to point out that the "IS-A" relationship also implies hierarchy of the concepts, meaning that all attributes from the more general concept are inherited to the more specific one, i.e. the child concept. As illustrated in Figure 2.1, a *carnivore* has *teeth* and a *cat* concept inherits the properties from the *carnivore*, which fact yields the conclusion that a *cat* also has *teeth*. In addition, a *cat* has a *breed*, e.g. American Longhair, Bengal, Birman, etc.

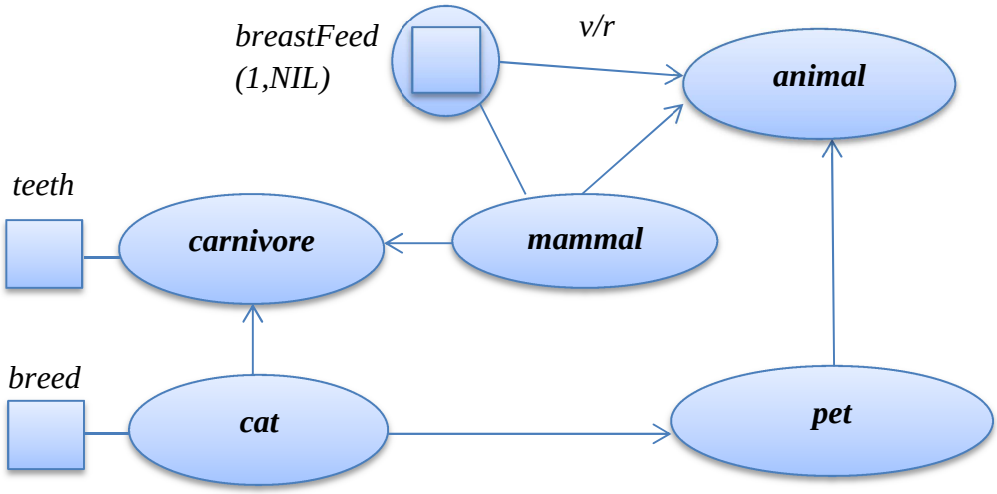

Figure 2.1 An Example of Network-based Representation Structure

The *breastFeed* node represents a role in the terminology, illustrated in Figure 2.1. This role has value and number restrictions, represented correspondingly by the labels *v/r*, meaning a value restriction, and *(1, NIL)*, which stands for the number of minimum and maximum breastfed children. The *NIL* symbol stands for infinity. This role description can be translated to natural language as "*A mammal is an animal, who can breastfeed at least one child and all children are also animals*".

It is important to point out that there could be also implicit relationships among the nodes of the network and this is one of the major tasks of the knowledge representations systems, namely to identify and take into account these implicit relations between the concepts during the reasoning process. For instance, as observed in Figure 2.1. the concept *cat* and *carnivore* are explicitly connected, same as *mammal* and *carnivore*, so it can be concluded that a *cat* is also a *mammal* even though there is no direct link between *cat* and *mammal*. As a consequence, the main problem of the network-based representation structures becomes obvious when the complexity and number of the concepts relationships increases, since it becomes more difficult to precisely characterize, recognize and compute all the relations among concepts from the terminology [1]. A solution to this problem is to define a formal language that provides both an accurate characterization and interpretation of the meaning of the network and its elements.

2. Fundamentals and Background

A lot of the ideas and principles, implemented in the first semantic networks and frame-based systems, have been the key to the successful development of the KL-ONE knowledge representation systems, which consequently sets the fundamentals of the logic-based characterization formalism. The language of KL-ONE explicitly represents and provides a logical basis for interpreting conceptual and role information, as well as the notion of subsumption, conjuction, role hierarchies, etc. As a consequence, a precise and distinct semantics was provided for the KL-ONE system, which resulted in the first Description Logics definition [1].

2.2 Description Logics

The knowledge of the application domain or the "world" in the Description Logics is represented by first defining all relevant domain concepts, namely the domain terminology. After that the properties of characteristic domain individuals or objects are specified, based on the previously defined concepts, thus creating the description of the application domain. The main difference of Description Logics in comparison to some of its predecessors is the fact that it is equipped with a formal, logic-based semantics. Another very important distinguishing characteristics of Description Logics is the central role of reasoning in this formalism, namely it is possible to infer implicit knowledge about the "world" from explicit facts included in the knowledge base [3].

2.2.1 Application and Reasoning

The application of Description Logics is very broad and it may be found in many intelligent systems for processing information, including natural language processing, database management, software engineering, digital libraries, web-based information systems and many others, because it supports useful and practical inference patterns to describe, classify and understand the human world. In fact, Description Logics facilitate the classification of concepts and individuals. Moreover, it not only specifies sub-concept/ super-concept relationships among different concepts, thus allowing subsumption, but it also provides the information whether a single individual or an object is an instance of a certain concept. This feature can in addition give important knowledge about the attributes or characteristics of a specific object. In fact, the subsumption is the main inference on concept expressions in Description Logics. In order to determine subsuption, it should be checked whether a specific concept B is more general than a concept A. The symbolic representation of subsumption in DL is typically written as $A \sqsubseteq B$ [1], namely B subsumes A or the concept B (the subsumer) is more general than the concept A (subsumee). Considering the example in Figure 2.1, it can be concluded that *pet* $\sqsubseteq$ *animal* and *mammal* $\sqsubseteq$ *animal*, meaning the concept *animal* subsumes or is more general than both the *pet* and the *mammal* concepts.

Another typical example of inference on concept expression in Description Logics is the concept satisfiability. The main task of checking satisfiability is to test whether the empty

concept is not necessary denoted by any concept expression. Concept satisfiability can be also regarded as a special case of the subsumption, where the subsumer is the empty concept, thus inferring that a specific concept is not satisfiable, i.e. contradictory [1].

In general, investigating the complexity of computing a specific inference problem in logics is a very important issue. This problem also occupies a fundamental part in the Description Logics field of research. Dating back to 1984, Brachman and Levesque used the simple structure of Description Logics in order to argue that there is a proportional relation between the expressive power of a knowledge representation language and the complexity of reasoning [4]. In other words, the more expressive the language is, the more difficult the reasoning is. This tradeoff between the expressivity of the Description Logics and the computational difficulty of its reasoning problems is one of the fundamental research topics in this area.

2.2.2 Syntax and Semantics

There are a lot of variations of the Description Logics, but the base description language is the AL-language (attributive language). The elementary descriptions are atomic roles and concepts. All other complex descriptions can be built with the help of these atomic constructors in an inductive way. It is commonly accepted that the abstract notation of atomic concepts is represented by the letters *A* and *B*, while the representation for atomic roles is *R*. The complex concept descriptions are denoted by the letters *C* and *D*. The main syntax rules of the AL, forming concept descriptions, are presented in Figure 2.2

C, D	$\rightarrow A \mid$	*(atomic concept)*
	$T \mid$	*(universal concept)*
	$\bot \mid$	*(bottom concept)*
	$\neg A \mid$	*(atomic negation)*
	$C \sqcap D \mid$	*(intersection)*
	$\forall R.C \mid$	*(value restriction)*
	$\exists R.T \mid$	*(limited existential quantification)*

Figure 2.2 AL Syntax [3]

It is important to point out that negation in AL is only applicable to atomic concepts. Furthermore, when using role description constructors only the top concept is allowed in the scope of existential quantification, as illustrated in Figure 2.2. Referring to Figure 2.1, where the description of an animal has been illustrated in a graphical way, this approach can be consequently extended and presented in terms of Description Logics. For instance, supposing that an *Animal* and a *Carnivore* are atomic concepts, then by using the intersection operator the complex concepts of meat-eating animals and plant-eating or herbivore animals are formally given by $Animal \sqcap Carnivore$ and $Animal \sqcap \neg Carnivore$. Besides, if supposing that *breastFeed* is an atomic role, then the concept of those animals, which are mammals, can be constructed as $Animal \sqcap \exists breastFeed.T$. All other non-mammals animals can be described as $Animal \sqcap \forall breastFeed.\bot$.

After presenting the base syntax of the AL, the formal semantics is consequently explained. In fact, the semantics of description logics is specified by defining the concepts as sets of individuals and respectively the roles as pairs of individuals, which are particularly connected to a specific domain. The most important fragments of the semantics are the interpretations and the interpretation function. In the interpretation function, a set in the form $A^I \subseteq \Delta^I$ is assigned to every atomic concept A and a binary relation $R^I \subseteq \Delta^I \text{x } \Delta^I$ is assigned to every atomic role R, where I stands for an interpretation function on a non-empty set Δ^I, i.e. the actual domain of interpretation [3]. The semantics of concepts and roles, which are not atomic, is realized with the help of recursive definitions, as illustrated in Figure 2.3. These definitions are similar to the ones, presented in Figure 2.2 .

$$
\begin{aligned}
T^I &= \Delta^I \\
\bot^I &= \emptyset \\
(\neg A)^I &= \Delta^I / A^I \quad \textit{complement, meaning negation} \\
(C \sqcap D)^I &= C^I \cap D^I \quad \textit{union, meaning disjunction} \\
(\forall R.C)^I &= \{ a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I \} \\
(\exists R.T)^I &= \{ a \in \Delta^I \mid \exists b.(a,b) \in R^I \}
\end{aligned}
$$

Figure 2.2 AL Semantics [3]

As previously mentioned, by extending the AL language, the expressive power of the Description Logics can be increased. For example, constructors as union of concepts, full existential quantification, number restrictions, negation of arbitrary concepts, etc. can be added in order to extend the AL, forming a new variation of the AL language.

2.2.3 TBox and ABox

Description Logics are very beneficial and practical in the design of knowledge-based systems since they provide a representation language in order to define a knowledge base and techniques to realize inference reasoning over this language. A very distinct differentiation of intensional and extensional knowledge can be noticed in the DL knowledge base. In the context of Description Logics, intensional knowledge stays for the information that is general about the problem domain and extensional knowledge represents the knowledge or information, specifying a particular problem. The knowledge base in DL consists of two main components- a TBox and an ABox. The TBox represents the intensional knowledge and it builds up the terminology. The TBox contains the vocabulary of the application domain and it contains definitions of concepts, roles and their general properties. A concept denotes a set of individuals, while a role represents a relationship among these individuals. On the contrary, the ABox represents the extensional knowledge or assertions about the individuals of the domain of interest in terms of the initially defined vocabulary [1, pp. 12-15].

There are two types of logical declarations or terminological axioms in the TBox – equivalences and inclusions. The logical equivalence provides both necessary and sufficient

conditions for classifying an individual. For instance, the concept definition of a *Man* in a TBox can be declared as the axiom, shown in Figure 2.3.

$$Man \equiv Person \sqcap Male$$

Figure 2.3 A Terminology with Equivalence Axiom

The form of definition of the concept *Man* of Figure 2.3 is much stronger than the one, illustrated in Figure 2.4, since the inclusion axiom of Figure 2.4 only imposes necessary condition, i.e. the concept is not defined completely. Inclusions generalize equivalences and therefore they are also called GCIs (General Inclusion Axioms). In practice they are a very convenient way to introduce new concepts or roles into a definite TBox, which terms could not be defined completely.

$$Male \sqsubset Person$$

Figure 2.4 A Terminology with Inclusion Axiom

Both equality and inclusion axioms not only allow concept definitions, but also roles. A TBox with concepts and roles within a family is presented in Figure 2.5.

$$\begin{aligned} Woman &\equiv Person \sqcap Female \\ Man &\equiv Person \sqcap \neg Woman \\ Mother &\equiv Woman \sqcap \exists hasChild.Person \\ Father &\equiv Man \sqcap \exists hasChild.Person \\ Parent &\equiv Mother \sqcup Father \end{aligned}$$

Figure 2.5 A Family Terminology Example [3]

The concept *Woman* from Figure 2.5 is defined to be a *Female Person*, a *Man* is a *Person*, who is not a *Woman*. In addition, a *Mother* is declared to be a *Woman*, who has children and a *Father* is specified to be a *Man* with children. Consequently, a *Parent* in the TBox of Figure 2.5 is classified as either a *Mother* or a *Father*.

Similar to the TBox, the ABox also deals with concept and roles, thus describing the current state of affairs, but what is the peculiar about the ABox is the fact, that in an ABox the individual plays a central role. In other words, individuals are introduced by assigning names to them and asserting their properties. It is a common practice that individuals are often labeled as *a*, *b* or *c* and by borrowing the notation of the TBox, assertions of the form *C(a)* and *R(b,c)* can be generated. An ABox is in fact a finite set of assertions. For instance, referring again to the animal example from Figure 2.1, if *TOM* and *JERRY* are individual names, then *Cat(TOM)* means that *TOM* belongs to the interpretation of the concept *Cat*, i.e. *TOM* is a cat and *Mammal(JERRY)* means that *JERRY* is a mammal. In addition, when taking into account individual role assertions, it can be concluded that *JERRY* suckles from *MARY* for the example of *breastFeed(MARY, JERRY)*.

2. Fundamentals and Background

There are two key inferences when considering a TBox and an ABox – satisfiability and consistency. Satisfiability of the TBox means that there is a model of the TBox, i.e. an interpretation of all axioms in the TBox, making all of them true. Taking into account an ABox, it is the case that an interpretation ***I*** satisfies the ABox ***A*** if it satisfies every single assertion in ***A***, thus ***I*** being the model of the assertion of the ABox. Combining, both satisfiabiliy definitions about the TBox and ABox, it can be concluded that interpretation "***I*** *satisfies an assertion* ***a*** *or an ABox* ***A*** *with respect to a TBox* ***T*** *if in addition to being a model of* ***a*** *or of* ***A****, it is a model of* ***T***" [3].

After checking the satisfiability, a typical task of a knowledge representation system is also to verify whether the representation of the particular knowledge is consistent. If that is not the case, then arbitrary or wrong conclusions can be drawn from a logical point of view. For instance, if the assertions *Father(PETER)* and *Mother(PETER)* are contained in a specific ABox, then it must be possible to detect that together with the TBox, presented in Figure 2.5, these statements are not consistent, since *Father* and *Mother* are interpreted as concepts, having disjoint extensions in the current example. However, if taking into account an empty TBox, the discussed assertions are consistent, because no restrictions of the interpretation of *Father* and *Mother* exist, so they may have a common element [3]. Both satisfiability tests of descriptions and consistency tests of sets of assertions are beneficial in order to determine whether a specific KB is in particular meaningful.

Another known technique for checking whether domain models are correct or to improve the optimization of queries, formulated as concepts, is the subsumtion, i.e. verifying whether a specific concept is more general than another concept. Furthermore, relationships such as disjointness, equivalence, etc. also play a major role and are of significant interest in the research area of Description Logics and reasoning about knowledge.

Last but not least, it is important to point out that the TBox and the ABox are the two typical main building components of the knowledge base of a knowledge representation system, based on a Description Logics. An example of the graphical representation of such a design [3, p. 50] is illustrated in Figure 2.6. In addition to the storage of assertions and terminologies, the system also implements numerous services and techniques as the previously described reasoning tasks for checking satisfiability, consistency, subsumption, etc.

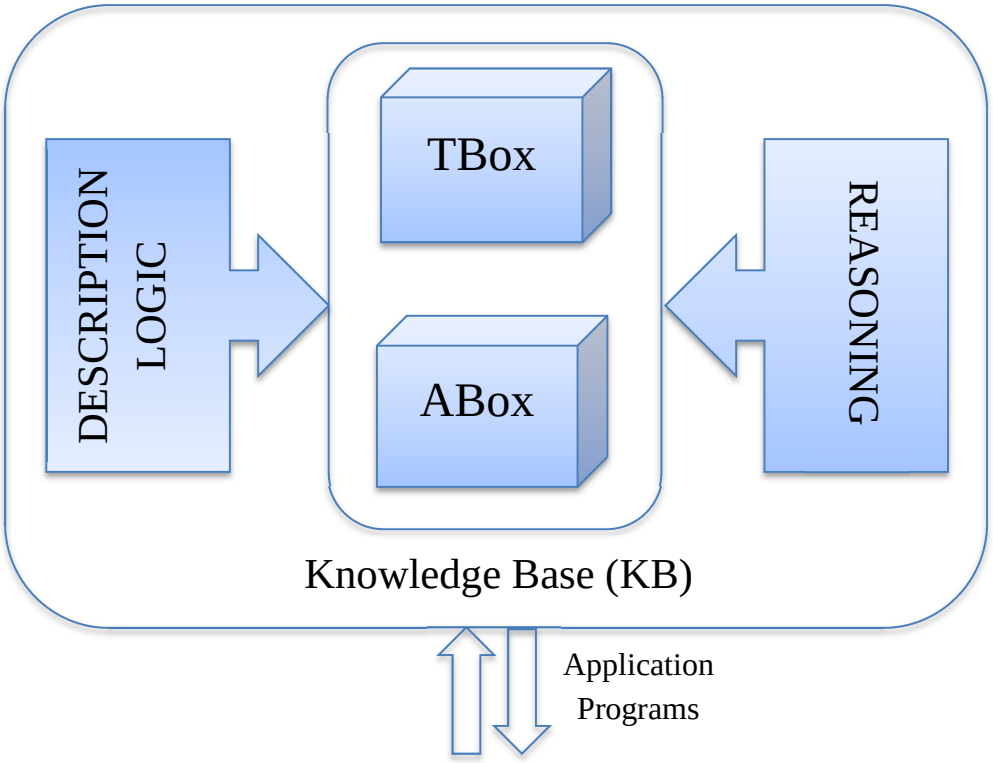

Figure 2.6 Architecture of a DL Knowledge Representation System

It is often the case that a TBox and an ABox are compared to relational databases and number of similarities among them can be found. For instance, in a simplified point of view it can be observed that a TBox is analogous to a database schema and an ABox can be treated as the actual data in the database. However, semantically these concepts differ in their essence. On the one hand, only one single interpretation is depicted by a database instance, while on the other hand, a lot of interpretations, i.e. models, are represented by an ABox. For example, if the only assertion about *BOBBY* is *hasParent(BOBBY, MIKE)*, then interpreting this in the database context yields the conclusion that BOBBY has only one parent and his name is *MIKE*. However, an *ABox* with such assertion only indicates that *MIKE* is a parent of *BOBBY* and nothing more. In fact, an ABox has many models. In some of these models, *MIKE* is the only parent, but in others *BOBBY* is not an orphan and he has a mother and a father.

Analyzing the previously described simple example, it is obvious that the lack of information in a database indicates negative information, while the nonexistence of specific information in an ABox is only interpreted as a lack of knowledge. In other words, the information in the database is generally referred as complete and the one in the ABox as incomplete. Consequently, the query answering and inferences in DLs are more complicated than query answering in databases, since an ABox could represent infinitely many models. Moreover, nontrivial reasoning techniques may be required, e.g. case analysis [3].

2.3 Ontology Based Data Access

Description Logics has been an essential area of knowledge representation for the last decades in order to build up the foundations for a structured representation of complex information, enriching this information with a formal semantics. Due to the logic-based formalism of the DL, an effective support for automated reasoning can be provided for solving various tasks and problems associated with data management.

Ontologies are considered as a suitable, flexible, powerful and efficient formal tool or approach that can deal with sophisticated data management tasks [5]. In fact, ontologies provide a representation schema, describing a formal conceptualization of a specific domain of interest. Its specification incorporates several levels. The core distinct layers are the intensional level, where the conceptual structure of the domain is specified, namely the TBox, and the extensional level, where instances of the conceptual elements from the intensional level are defined, namely the ABox. In addition, an ontology may also have an extra tier, i.e. a meta-level, where a set of modeling categories are being specified [6].

The proposal of using an ontology as a conceptual view over a repository is reasonable, but the key point is that the conceptual layer with the help of which the lower data layer is accessed, should not add a considerable overhead when processing the data [7]. The problem becomes more critical when dealing with large amount of data, i.e. in particular when considering geographical and spatial data. A graphical representation of a design of an information system, using ontologies as a technical tool for providing a conceptualization over a specific domain, is presented in Figure 2.7

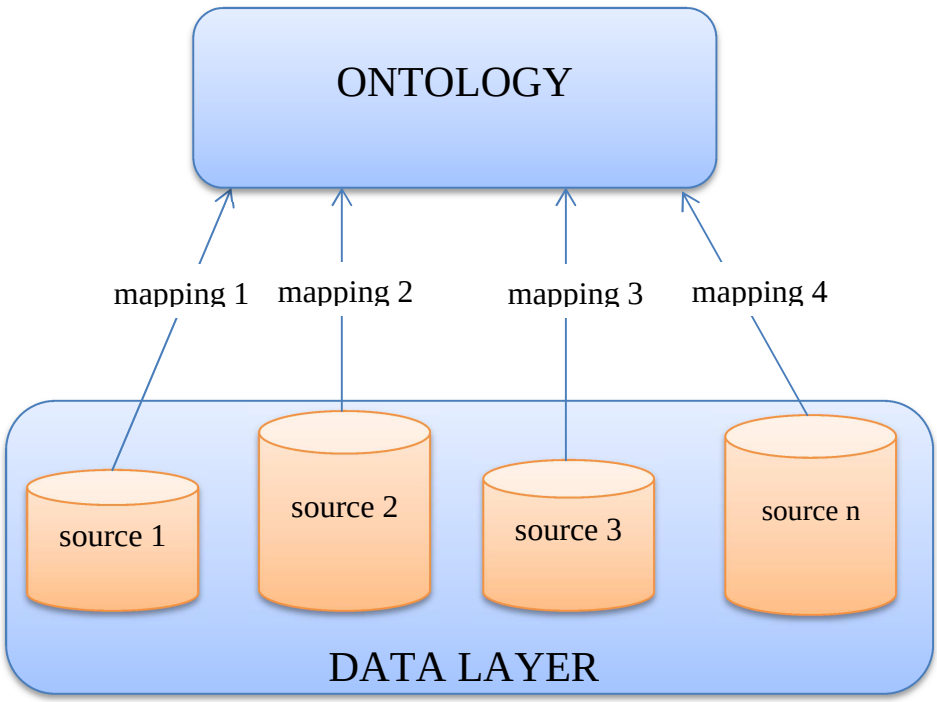

Figure 2.7 Design of an Information System, using Ontologies

The ontology in Figure 2.7 maps a specific domain of interest within an information system at a high level of abstraction. The relationship between the instances of concepts and roles in the ontology and the data at the sources are realized with the help of appropriate mappings. The advantage of this setting is the fact that it is not necessarily needed to be known how the data repository is organized and stored or where the data sources are located. In other words, the data sources are independent from the ontology and they are situated at different levels of abstraction. In addition, information systems, which are implemented with the help of ontologies can „communicate“ among each other by sharing information. This cooperation is performed at the level of the conceptualization, without the need of connecting the data sources, thus making the ontology also the core ingredient of cooperation among information systems.

The design scenario, illustrated in Figure 2.7 is a typical representative of the so called Ontology-Based Data Access (OBDA). The pre-existing data sources build up the data layer of the information system, on the top of this layer there is a conceptual view of the data, that is consequently to be seen by the user. The conceptual view is formed by the ontology, which is the only access point for the client to interact with the system. The purpose of the OBDA is to link a set of autonomously existent data to a specific ontology without being necessary to structure this data with the aim of saving the ontology instances [5]. As already mentioned the autonomous data and the ontology are at different abstraction levels and they may be specified in terms of not necessarily the same formalism. For instance, relational data models are usually used for the representation of data sources, while ontologies are expressed by logical languages, such as Description Logics and its successors. Considering these different characteristics Antonella Poggi [5, pp. 134-135] and her colleagues summarize the five most important issues, when dealing with the interaction between the data layer and the conceptual view of the domain of interest, i.e. the ontology:

I. The chosen ontology language for expressing complex semantic conditions at the intensional level should be expressive enough and at the same time its computational complexity of reasoning should be manageable as previously mentioned in Chapter 2.2.1.

II. In modern applications, the amount of information, stored in the Data Layer, may be very large. Because of this reason an appropriate technology should be used that is able to handle large qualities of data in an efficient and consistent way. Such a technological solution, fulfilling the requirement of effective data manipulation, is for instance a relational Database Management System (DBMS). Here it is important to find the balance how much the expressive power of the ontology formalism can be used in order to still make use of the effectiveness, simplicity and convenience of the query answering over relational DBMS.

III. A mapping approach should be implemented, which realizes the formal linking or translation of the stored data and the ontology. In other words, a technique should be found out that reconstructs the meaning of the source data in terms of the conceptual layer. This extra mapping layer is needed, because the data at the Data Layer is stored independently of the ontology.

IV. The way data is expressed in a relational database differs from the way the corresponding information is provided in an ontology. A mismatch exists, since the main components managed by the data sources are stored values, i.e. data, and the elements managed by the ontology are abstract objects, which are instances of

concepts and roles in the ontology. This issue is widely known in the literature as the impedance mismatch problem and the mapping approach, described in the previous point, should cope with this problem.

V. The last issue when building OBDA systems is the necessity of implementing a query answering method. The aim of this method is to facilitate the process of reasoning at the conceptual level and at the same time to provide mechanisms for efficient data access at the source by incorporating the mapping approach. In other words, the OBDA system should be able to translate any client request into an appropriate query that is consequently posed to the source.

2.4 Spatial Databases and Existing OBDA Software Tools

In OBDA the extensional knowledge in the database is extended by intensional knowledge in the form of an ontology. In a nutshell, the ontology builds up the conceptual view of an information system over a repository, as illustrated in Figure 2.7. Typically a geographical information system with ontologies, considering GEO-thematic, topological or spatial orientation scenarios, should encapsulate backend capabilities in order to store the large amount of GIS data. The most popular ways of building storage environments in general are file systems and databases. A file system is part of every operating system and its responsibilities are to manage and store computer files and data on storage devices. Depending on the way how records are saved or retrieved within a file, the files are specified into three main types, namely sequential, index sequential and direct access based. Nevertheless, a more reasonable and more practical possibility for storing GIS data is to use a database rather than a file system. The main benefit of this suggestion is the fact that for systems, processing a large amount of GIS data, a database is more appropriate due to its quick data access, compact data storage and standardized querying technologies. There are numerous spatial extensions and databases on the market today. The most popular of them are:

- PostgreSQL, which is an open source object-relational database system. It uses the spatial extension PostGIS to provide support for geographic objects and corresponding spatial and geometric functions [8].

- Oracle Spatial provides an SQL schema and features for facilitating the storage, update and retrieval of spatial data in an Oracle database [9].

- Microsoft SQL Server supports the geography and geometry data types for managing spatial data since version 2008 [10].

- IBM DB2 Spatial Extender together with the Geodetic Data Management Feature offer support for spatial types, providing spatial and geodetic capabilities in order to query, maintain and create spatial data [11].

2. Fundamentals and Background

The open source PostgreSQL database management system together with the spatial extension PostGIS turn out to be the most appropriate option for storing persistent GIS information, taking into account license issues, performance, stability and possibility to handle geographic data in a standardized way. Moreover, the plugin PostGIS Shapefile and DBF Loader allows importing of GIS data, encapsulated in shapefiles, directly into the database. For instance, TIGER/Line® Shapefiles and TIGER/Line® Files can be loaded automatically in the database. These files are distributed to the public free of charge and they represent spatial extracts from the Census Bureau's MAF/TIGER® (Master Address File/Topologically Integrated Geographic Encoding and Referencing) database [12]. They contain geographic information such as roads, rivers, railroads, hospitals, etc. as well as other geographic data and regions. The TIGER/Line® Shapefiles are very beneficial from usability and practical point of view when building and later testing a geographical information system with ontologies, since they provide up-to-date and real-world GIS data.

Currently there is no commercial system with ontologies that simultaneously combines reasoning over a spatial and a thematic domain, using a Perfect Rewriting approach. However, related work, scientific projects and system prototypes have been broadly investigated and developed in the last years. For instance, FaCT [13], D_{LP} [14], RACER [15], QuOnto [16] are systems, dealing with expressive Description Logics and OBDA, but they do not incorporate facilities for space representation and reasoning. The innovative OnGIS [17] system is based on OWL ontologies and it provides limited ontology driven geospatial search and integration. The advantage of this system is that it can spatially relate search results. However, its expressivity is currently very limited, since it provides only two spatial restrictions, i.e. "inside" and "distance" restrictions. In addition, non-spatial data is not supported and significant performance issues and computationally expensive spatial operations generate further problems, which altogether do not classify the OnGIS prototype and the former described applications as reliable ontology-based GIS systems, which guarantee a complete, correct, and efficient query answering mechanism. That is the reason why, the development of such a system is beneficial in the fields of Description Logics and other practical areas of science and life, such as city planning, urban management, construction of eco systems in forestry, etc. The next chapter investigates formal techniques, approaches, issues, and technologies relevant for the implementation of a stable ontology-based GIS system for query answering.

3 Analysis

This chapter presents a problem and requirement analysis. It starts with a detailed description of the issues of OBDA, discussing the trade-off between expressive power and computational complexity of reasoning, as well as different approaches and resulting difficulties how to link data to ontologies. Then the core ideas of DL-Lite and several DL-Lite extensions are presented. Consequently, the main part of this chapter discusses the DL-Lite Combined Geo-thematic Logics and GCQ^{+}. Moreover, two approaches for ontology based query answering over spatial databases are evaluated and the Adapted Perfect Rewriting Algorithm is explained in details. Finally, the chapter ends with a list of all program and technical requirements, based on the previously analyzed concepts, technologies, solution proposals, and limitations.

3.1 Expressive Power versus Efficiency of Reasoning

Before going into details of investigating the first issue when dealing with OBDA, i.e. expressive power versus efficiency of reasoning in the scenarios, where large quantities of data is to be accessed, it is crucial to first present a number of basic notions and terms related to computation complexity. When analyzing or measuring the computational complexity of reasoning in an OBDA system or a query language, the critical term of interest is the data complexity or the complexity with which a query is evaluated as a function of the size of the data or the database [18], discarding the size of the TBox and the query. In general, a computational problem can be specified by a corresponding complexity classes.

A complexity class is defined by its models of computation, the bounded resources and the corresponding bounds. Examples of models of computation are deterministic and nondeterministic Turing machines, Boolean circuits, etc., and examples of resource constrains are logarithmic time, polynomial space, etc. [19]. The relationship among some of the most popular complexity classes is presented in Figure 3.1 [7], where it can be seen that AC is contained in LogSpace and LogSpace is contained in NLogSpace, etc. A representative example of a problem with a complexity AC^0 is the evaluation of first-order queries (e.g. SQL queries) over a relational database [20], which justifies the statement that a relational DBMS handles large amount of data in an efficient way.

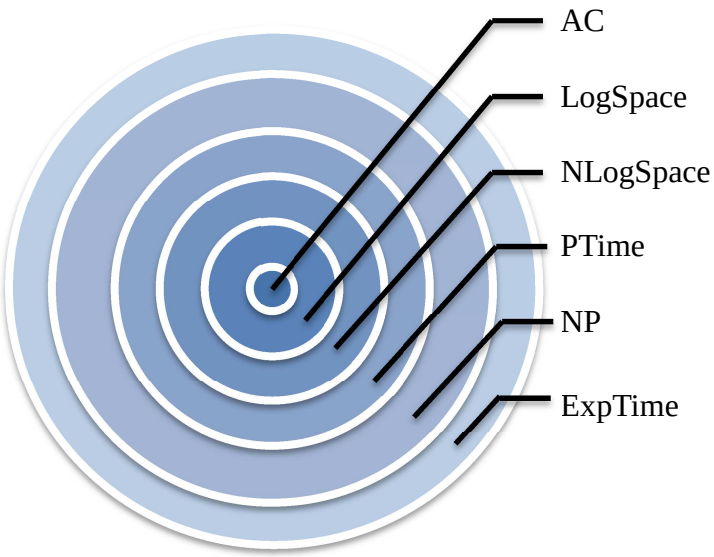

Figure 3.1 Complexity Classes Relationship

Considering the first issue of OBDA with respect to large quantities of data, several attempts can be found in the literature, which try to overcome the problem of choosing such a language that is simultaneously expressive enough and ensures that reasoning is still computationally tractable, referring to data complexity. In this context tractability means *"solvable by a polynomial time algorithm"* [7]. In other words, it can be accepted that reasoning in the intensional part of the ontology, i.e. the TBox, is exponential, but reasoning in the data must be at least in NC or in AC. According to [5] the OWL-DL and OWL-Lite, which are sublanguages of the Web Ontology Language and endorsed by the World Wide Web Consortium (W3C), are coNP-hard, thus classifying these languages as inappropriate solutions to the first OBDA issue (cf. Chapter 2.3). However, several proposals such as Horn-SHIQ [21], EL++ and DLP provide polynomial reasoning, thus making them attractive expressive Description Logics. A number of variations and sublanguages of the DL-Lite family are also investigated and currently seen as possible solutions, since the DL-Lite Logics (in particular DL-Lite$_A$ [5]) guarantee complex query answering in AC^0 with respect to data complexity. Besides, the more important advantage of the DL-Lite (in particular DL-Lite$_{A,id}$ [7]) is that after an initial reformulation procedure (cf. Chapter 3.5.1), not dependent on the amount of the data, a SQL query can be generated out of the original query, thus delegating the query processing to the RDBMS, i.e. having a data complexity in AC^0. This feature of the DL-Lite also addresses the second issue of OBDA. More details about this method are presented in Chapter 3.5

3.2 Linking Data to Ontologies

In the traditional environment of the Description Logics, it is adopted that the data is completely maintained in the ABox of the ontology [22] and the ABox is capable of being used with the TBox without any modifications. This means that there is compatibility between

the extensional and insentional levels of the ontology. In fact, the TBox uses the lexicon of roles, concepts and attributes and the ABox contains contingent facts, i.e. concept assertions or role assertions. Depending on specific requirements, the physical storage of the ABox can be either maintained in the main memory of the DL reasoner, or in a secondary storage, which process is again delegated by the reasoner itself.

Nevertheless, in real world settings and in particular in geographic scenarios, where either the ABox is very large, there is no direct control of the data, since it belongs to some other organizations, or the data is stored in different data sources, then a relational database is needed. This requirement however poses the issue of the third and fourth OBDA problems, introduced in Chapter 2.3. On the one hand, the source data in the database is stored in terms of values of strings, integers, dates, etc. On the other hand, the instances of concepts and roles in the ABox are abstract objects, i.e. the objects are not materialized and the ABox is thus considered as virtual. A possible solution to this scenario is to specify mappings between the data source and the ontology, as graphically illustrated in Figure 3.4. Such a mechanism is proposed in [5] and [22], enabling linking of existing data sources to an ontology expressed in an extension of the DL-Lite Logic. The fundamental idea of this approach is that every mapping assertion consists of two mutually associated parts - a query, the aim of which is to retrieve specific values from the database, and a set of atoms, specified in the vocabulary of the ontology (cf. Figure 3.4). With the help of Skolem functions, the transformation of data values into abstract objects is possible. Skolem functions output uniquely defined values for their arguments and are also used in XML schema mappings under the settings of XML data exchange scenarios [23]. Moreover, the objects are denoted by an ad hoc identifier, namely a term, obeying the unique name assumption (UNA) on terms, i.e. distinct individual terms denote distinct objects. The example in Figure 3.2, where a graphical representation of the relationship between a student and a lecture within a university is shown, can be used to illustrate this approach.

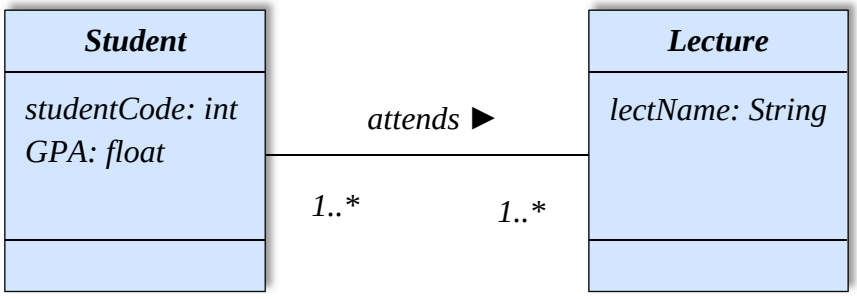

Figure 3.2 Graphical Representation of the Relation
between a Student and a Lecture

From Figure 3.2 and the multiplicity *1..** , it can be concluded that a *Lecture* is attended at least by one *Student*, and students can attend several lectures. Furthermore, analyzing Figure 3.3, it is of a peculiar interest to point out that the actual data is stored in a database, containing the tables *University*, *StudentGrade* and *Student*, where a student is identified by his matriculation number and a lecture is identified by its name. As a result, the abstract

object *student* should be created out of his *MatrNum*, namely *std(MatrNum)* and the lecture should be created by its name, i.e. *lect(LecName)*.

University [MatrNum: Varchar; LecName: Varchar]
Students and lectures they attend

StudentGrades [Code: Varchar; GPA: Numeric]
Student's code and corresponding Grade Point Average (GPA)

Students[Code: Varchar; MatrNum: Varchar]
Student's code with a student matriculation number

Figure 3.3 Table Signatures of a Sample Database

In order to create the object identifiers and also address the impedance mismatch problem, a set of function symbols *{std, lect}* is introduced. Every function symbol has an associated arity, which aim is to specify the number of the accepted arguments. Besides that, it is assumed that the data, stored in the relational database, is denoted by value constants and the objects, managed by the ontology, are denoted as object terms. These terms consist of function symbols and value constants. For instance, if a *student* is identified by a *matriculation number* and *31897* is a matriculation number, stored in the database, then the object term *std(31897)* denotes a *student*.

The proposed mappings assertions between the database and the TBox have the following formulation: an *SQL query ~> a set of atoms / a CQ over the TBox*. A conjunctive query (CQ) in DL-Lite$_A$ is a statement of the form $q(\mathbf{x}) \rightarrow conj(\mathbf{x},\mathbf{y})$, where $q(\mathbf{x})$ is the head and $conj(\mathbf{x},\mathbf{y})$ is the body of the query, representing a conjunction of atoms. The variables occurring in the head of the query are called distinguished variables, i.e. the tuple $\boldsymbol{x}$, while $\boldsymbol{y}$ is a tuple of distinct existentially quantified variables, which do not occur in **x** [5] , namely non-distinguished variables**.** If a variable occurs more than once in the body of a query, then it is a shared variable. Besides, if a shared variable, a constant or a distinguished variable is an argument in a query atom, then it is called bound. Correspondingly an unbound argument corresponds to a non-distinguished non-shared variable, marked symbolically as "_". It is important to point out that the set of atoms over the TBox in every mapping assertion should involve only distinguished variables, which respectively may include variable terms with function symbols. A sample mapping of the example presented in Figure 3.2 and 3.3 is illustrated in Figure 3.4.

M1: *SELECT MatrNum, LecName from University* ~> *Student(std(MatrNum)), ATTENDS(std(MatrNum), lect(LecName)), lectureName(lect(LecName), LecName)*

M_{left} — SQL query over the database

M_{right} — Conjunction of atoms over the TBox

Figure 3.4 Sample Mapping Assertion

The data-to-ontology mapping in Figure 3.4 maps every tuple *(m, l)* in table *University* to a student *std(m)*, who attends lecture *lect(l)* with name *l*.

In order to summarize the method, proposed by [5] and [22], it can be concluded that mapping assertions are of crucial importance for the construction of an OBDA system. Moreover, such a system (i.e. an ontology with mappings) can be symbolically defined as a triple $O = < T, D, M >$, built up of a TBox *T*, relational database *D* and a number of mapping assertions *M* , used to bridge the "gap" between the TBox and the database. In addition, the mappings *M* can be subdivided into two classes – typing mapping assertions and object-to-data mapping assertions. The typing assertions are useful in order to match the appropriate types of the data values from the database and the types, specified in the ontology. Referring again to the University example from Figure 3.2 and 3.3, it could be also specified in the TBox that the attribute *lectureName* is of type string, i.e. *lectureName* $\sqsubseteq$ *xsd:string*. The corresponding typing mapping is shown in Figure 3.5.

M^{type}: SELECT LecName from University ~> xsd:string

Figure 3.5 Sample Typing Mapping Assertion

Considering the definition of an ontology, namely $O = < T, D, M >$, the split version of the ontology is $O = < T, D, split(M) >$, where *split(M)* is a new group of mapping assertions. The *split(M)* consists of all typing assertions and mapping assertions. The peculiarity is that a new mapping assertion is added for every atom of the right part of the mapping itself, namely M_{right} in Figure 3.4. When this rule is applied to the mapping assertions from Figure 3.4, a group of split mapping assertions is generated, illustrated in Figure 3.6. The ontology with mappings and its corresponding split versions are logically equivalent [7], meaning that an ontology with mappings can be reduced to its split equivalent. In addition, the computation of the reduction has PTime complexity with respect to the size of the mappings for DL-$Lite_A$ ontologies [5] and LogSpace complexity for DL-$Lite_{A,id}$ [7]. In both cases, the reduction is independent of the size of the data.

M_{11}: SELECT MatrNum, LecName from University ~> Student(std(MatrNum))

M_{12}: SELECT MatrNum, LecName from University ~> ATTENDS(std(MatrNum), lect(LecName))

M_{13}: SELECT MatrNum, LecName from University ~> lectureName(lect(LecName), LecName)

Figure 3.6 Split Version of a Mapping Assertion

There are also other known approaches when dealing with the issue of linking data to ontologies. For instance, a Relational to Ontology language (R_2O) is proposed in [24], which

main characteristics are its extensible and declarative nature. With the help of this language various mappings between ontologies in RDF (Resource Description Framework) and relational database schemas can be created. The usage of R_2O is concentrated in the context of the Semantic Web. This language is expressive enough to cope with complex mappings, but however it does not address directly the impedance mismatch problem [5]. Another approach, presented in [25], but still having several of the disadvantage of the R_2O is the CARIN logical formalism, exploited in an information integration system, called PICSEL. In its essence, the CARIN approach resembles the detailed presented technique at the beginning of this chapter, namely the method for linking of existing data sources to an ontology expressed in *DL*-Lite$_A$.

3.3 DL-Lite

The DL-Lite family is part of the Description Logics family and the alphabet of DL-Lite also consists of symbols for atomic concepts and roles, value-domains, atomic attributes and constants. The peculiarity of the DL-Lite is the fact that it is not only logics, tailored to capture basic ontology languages and popular modeling formalism, but moreover query answering in DL-Lite is managed in an efficient way. This is achieved by keeping the complexity of reasoning low, taking advantage of the query optimization techniques in relational databases and relying on several rewriting algorithms and procedures. Other benefits of the DL-Lite are its possibility to also capture basic conceptual data models and object-oriented formalism, such as simple class diagrams [7], designed in the Unified Modeling Language (UML).

3.3.1 DL-Lite$_{core}$

There are currently numerous extensions and proposal variations of the DL-Lite family. Nevertheless, the basic one is specified as DL-Lite$_{core}$ [26]. The fundamental features of the DL-Lite$_{core}$ are that it allows for expressing:

- ISA assertions on concepts ($A1 \sqsubseteq A2$). For instance, subsumption can be realized, the concept *Pupil* is subsumed by the concept *Person*, i.e. $Pupil \sqsubseteq Person$;
- Disjointness of concepts ($A1 \sqsubseteq \neg A2$), namely the concept *Pupil* is not a *School*, i.e. $Pupil \sqsubseteq \neg School$;
- Role-typing ($\exists R^- \sqsubseteq A2$, $\exists R \sqsubseteq A1$), specifying that one of the components in a role is an instance of a specific concept. For instance, $\exists TEACHES\_TO^- \sqsubseteq Pupil$, meaning that the second component of the role *TEACHES_TO* is an instance of the concept *Pupil* or respectively the first component of the relation *TEACHES_TO* is an instance of a *Teacher*, i.e. $\exists TEACHES\_TO \sqsubseteq Teacher$;
- Mandatory participation ($A1 \sqsubseteq \exists R$, $A2 \sqsubseteq \exists R^-$) and non-participation constants ($A1 \sqsubseteq \neg\exists R$, $A2 \sqsubseteq \neg\exists R^-$), stating correspondingly that all instances of a concept either participate or do not participate in a role as a first or respectively second

component. For example, from the assertions *Teacher* ⊑ ∃*TEACHES_TO* and *Pupil* ⊑ ∃*TEACHES_TO* ¯, it can be concluded that all teachers teach all students.

3.3.2 DL-Lite Extensions

Apart from the DL-Lite$_{core}$ logic, there are several DL-Lite sub-families. The first representative is the DL-Lite$_{A.}$ Its main peculiarity in comparison to other logics is the fact, that it concentrates on the distinct differentiation between values and objects [5]. It identifies concepts as abstraction for objects, thus distinguishing them from value domains, specifying concrete data values. Furthermore, there is also a strict separation of attributes and roles, since roles stand for relations among objects, while concept attributes represent relation between objects and values. The TBox in DL-Lite$_A$ may contain intensional assertions of two types, namely inclusion assertions (concept, role, value-domain and attribute inclusion assertions) and functionality assertions (role and attribute functionality assertions). Functional assertions express global functionality of a role or attribute, e.g. if the role *TEACHES_TO* ¯ is defined as functional, i.e. *(funct TEACHES_TO* ¯*)*, this means that a *Pupil* may be taught at most by one *Teacher*. Besides, an inclusion assertion can be further split into two sub-groups, namely positive inclusions (PI) and negative inclusion (NI). A positive inclusion assertion is the assertion, that does not contain the complement/negation symbol " ¬ " on its right-hand side.

Another example for an extension of the DL-Lite family is the DL-Lite$_{A,id}$ that also provides identification constrains in addition to all features of DL-Lite$_A$ [7]. This identification constrains are based on paths [27]. The syntax, used to build up a path, is illustrated in Figure 3.7

$$\pi \text{ -> } R \mid D? \mid \pi_1 \circ \pi_2$$

Figure 3.7. Syntax of a Path

R in Figure 3.7 stands for an atomic role, the inverse of an atomic role, an attribute or the inverse of an attribute. *D* denotes a concept or a value-domain and *D?*, called a test relation, specifies the identity relation on instances of *D*, thus imposing that a path is closely connected to a certain concept. For instance, the test relation *HAS-PARENT* ∘ *Man?* is interpreted as the path that connects somebody to her or his father. Last but not least, $\pi 1 \circ \pi 2$ denotes the composition of path $\pi 1$ and path $\pi 2$. The composition of paths is similar to the definition of composition of functions in mathematics, because it is also a method of creating a new path (respectively relation) $\pi 1 \circ \pi 2$ from two given paths. The general definition of composition is presented both symbolically and graphically in Figure 3.8, where it can be seen that in order to reach y from x, two steps should be performed, namely from x to z, related to π_2, and from z to y, related to π_1.

$$\pi_1 \circ \pi_2 = \{ (x,y) \mid \exists z.\ \pi_2(x,z) \wedge \pi_1(z,y) \}$$

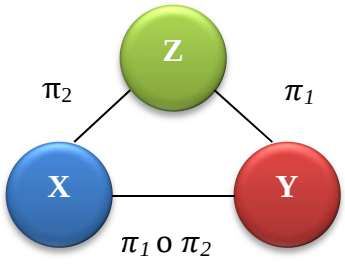

Figure 3.8 Composition of $\pi_1 \circ \pi_2$

After explaining the notion of paths, denoting complex properties for concept instances, the term identification constrain (also called an identification assertion) can be investigated into details. Every identification assertion has the form, illustrated in Figure 3.9

$$(id\ B\ \pi_{1,\ldots,}\pi_n)$$

Figure 3.9. Identification Assertion [27]

The basic concept B from Figure 3.9 is part of the syntax of an identification assertion and the $\pi_{1,\ldots,}\pi_n$ are component paths for $n \geq 1$. For instance, the identification constrain *(id Student HAS-MatrNum)* states that a student is identified by his matriculation number and there exist no other student, who has the same matriculation number.

The TBox in DL-Lite$_{A,id}$ may contain intensional assertions of three types, namely inclusion assertions, functionality assertions and identification assertions. Both the ABox in DL-Lite$_{A,id}$ and DL-Lite$_A$ are built up by membership assertions [7], which aim is to specify instances of concepts, roles and attributes. These assertions are described symbolically in Figure 3.10, where *A* represent the set of atomic concepts, *R* represents the set of atomic roles and *U* represents the set of atomic attributes. In addition, *o*, o_1 and o_2 are constant symbols for objects, while *v* is a constant symbol for a value.

$$A(o) \qquad P(o_1,o_2) \qquad U(o,v)$$

Figure 3.10. Membership Assertions in an DL-Lite$_A$ / DL-Lite$_{A,id}$ ABox

Examples of not as expressive extensions as DL-Lite$_{A,id}$ are DL-Lite$_f$, adding the potential to express functionality restrictions on roles and DL-Lite$_r$, adding disjointness and ISA assertions on roles [26]. These extra features make the DL-Lite$_f$ and DL-Lite$_r$ very appropriate to capture the main basic notions in the field of ontology languages, conceptual modeling formalisms and object-oriented models. In addition, the DL-Lite$_f$ and DL-Lite$_r$ are PTime in the size of the TBox, LogSpace in the size of the ABox and NP-complete in combined complexity, i.e. the total complexity dependent on the size of the TBox, ABox and the query.

The common feature of the previously discussed logics, being part of the DL-Lite family, is the fact that a TBox, encapsulating general properties of concepts and roles, and an ABox,

specifying instances of these concepts and roles, are the two separate building blocks of the knowledge base. Moreover, several extensions to the DL-Lite family can efficiently handle query answering over large amounts of instances, since the complexity of reasoning is considerably low. This is described in details in [26], where its is shown that the basic reasoning tasks such as computing subsumtion among concepts and roles and checking satisfiability of the entire knowledge base are polynomial in the size of the TBox and query answering is AC^0 in data complexity. Unfortunately, when trying to increase the expressive power of the language beyond that of the DL-Lite$_A$, DL-Lite$_f$ or DL-Lite$_{R,}$ then the data complexity of query answering increases rapidly to NLogSpace, PTime and coNP [28] .

As already discussed, the data in Ontology-Based Data Access is very large and it dominates the size of the intensional level of the ontology, i.e. the TBox. The situation becomes even more problematic when the data has geographical originin and a Geographical Information System over spatio-thematic ontologies should be build, since the geographical data consumes large space resources. In this scenario, the size of the TBox is negligible with respect to the size of the data, namely the ABox, so one of the most important measuring parameter to be taken into account is the data complexity. Although it can be accepted that reasoning is exponential on the intensional level, it is of crucial importance that reasoning in the data must be at least polynomial and even in a lower complexity class, i.e. the reasoning must be tractable. For instance, a quadratic dependence on the size of a large database can be also fatal.

A very important and beneficial property of several of the DL-Lite family extensions is the fact that they allow for first order logic (FOL) rewritability of both satisfiability checking and query answering. In a nutshell, these inference reasoning problems can be reduced to evaluating a FOL query over an ABox *A*, considered as a relational database (cf. Chapter 3.5). This database instance is known in the literature as *DB(A)* and it is interpreted as a minimal model of the ABox *A*. In the cases, when the data complexity is beyond AC^0, then the problem can be proved to be not FOL-rewritable [7, p. 319]. As a result, the positive aspects of current relational DBMS could not be used, since more powerful query answering engines are needed in the case when FOL-rewritability is not provided.

In order to decide on an appropriate DL language, being able to guarantee a computational feasibility with respect to query answering and at the same time to provide a sufficient expressiveness to capture spatio-thematic ontologies, the data complexity of the previously discussed logics of the DL-Lite family should be compared. This comparison is illustrated in Figure 3.11, from where it can be concluded that the most appropriate candidates, when taking into account the data complexity of query answering, are DL-Lite$_{A,id}$, DL-Lite$_A$, DL-Lite$_{core}$, DL-Lite$_{f,}$ DL-Lite$_r$, since all of them lie within the area surrounded by a yellow circuit, symbolizing the tractability border. Nonetheless, the DL-Lite extension DL-Lite$_{A,id}$ is the most expressive from the listed logics extensions. That is the reason why the DL-Lite$_{A,id}$ formalism can be referred as reasonable candidates for capturing ontologies in an efficient way. However on the other hand, the DL-Lite$_{A,id}$ logics is unfortunately not expressive enough to sufficiently model GIS data. Therefore, a further modified logic DL-Lite(RCC8) [29] of the DL-Lite family is proposed in order to overcome this particular issue.

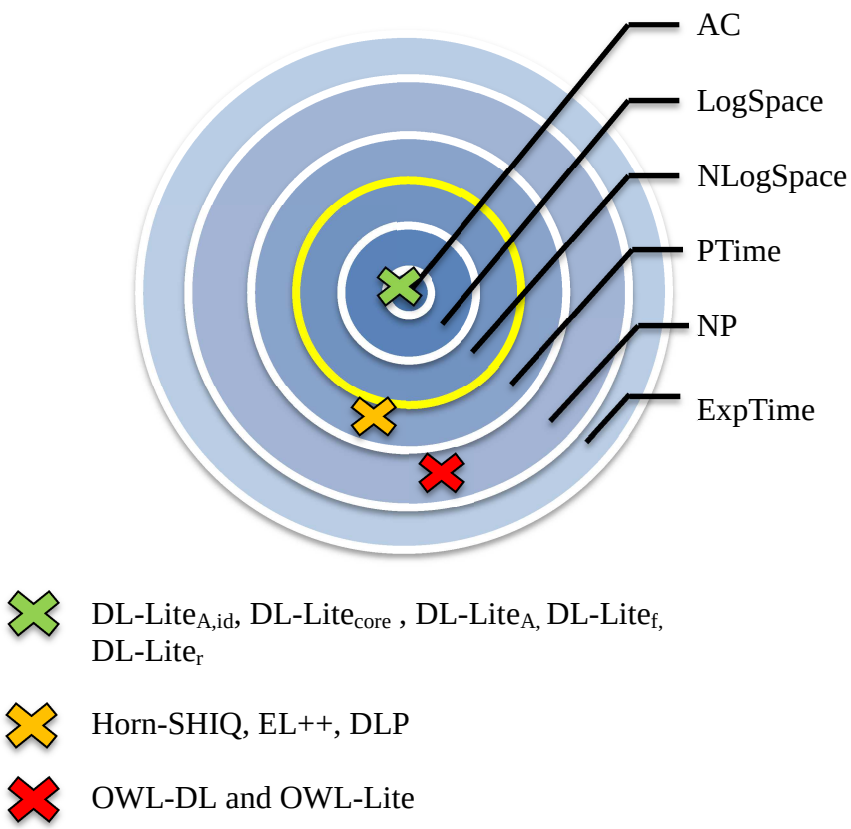

Figure 3.11 Data Complexity of Query Answering
for some Ontology-Based Languages

3.4 DL-Lite Combined Geo-thematic Logics and GCQ^{+}

Geo-thematic Logics should be capable of providing sufficient expressivity in order to describe spatial and regional configuration of objects. Moreover, these Logics should incorporate an expressive querying language allowing for First Order Query rewritability of query answering over spatial ontologies. A reasonable solution to these issues is advocated by the proposal of *"a weak coupling of DL-Lite with the expressive Region Connection Calculus RCC8"* [30, pp. 1,5-6] under the condition that the ABox is spatially complete. Nonetheless, before going into details aiming to explain the proposed DL-Lite(RCC8) formalism and GCQ+ query language, a short introduction of the Region Connection Calculus is needed in order to understand the main features and concepts of the DL-Lite(RCC8).

3.4.1 Region Connection Calculus

The Region Connection Calculus is family of spatial logics, developed to be used in order to represent spatial knowledge and reason about space, being one of the most widely used formalisms for qualitative spatial reasoning. The RCC is based on regions and the primitive

connectedness relation [31]. A primitive relation in the form *C(x,y)* is the basic building block of the RCC theory. This relation is defined on regions and it is interpreted as the region *x* is connected with the region *y*. Moreover, from an axiomatic point of view it is qualified as reflexive and symmetric. Additional eight basic relations are specified, using the representation of the primitive relation. These binary relations define base relations between regions and are the foundation of the RCC8 constrained language. The notation and the base RCC8 relations and their topological interpretation are presented in Figure 3.12 [32].

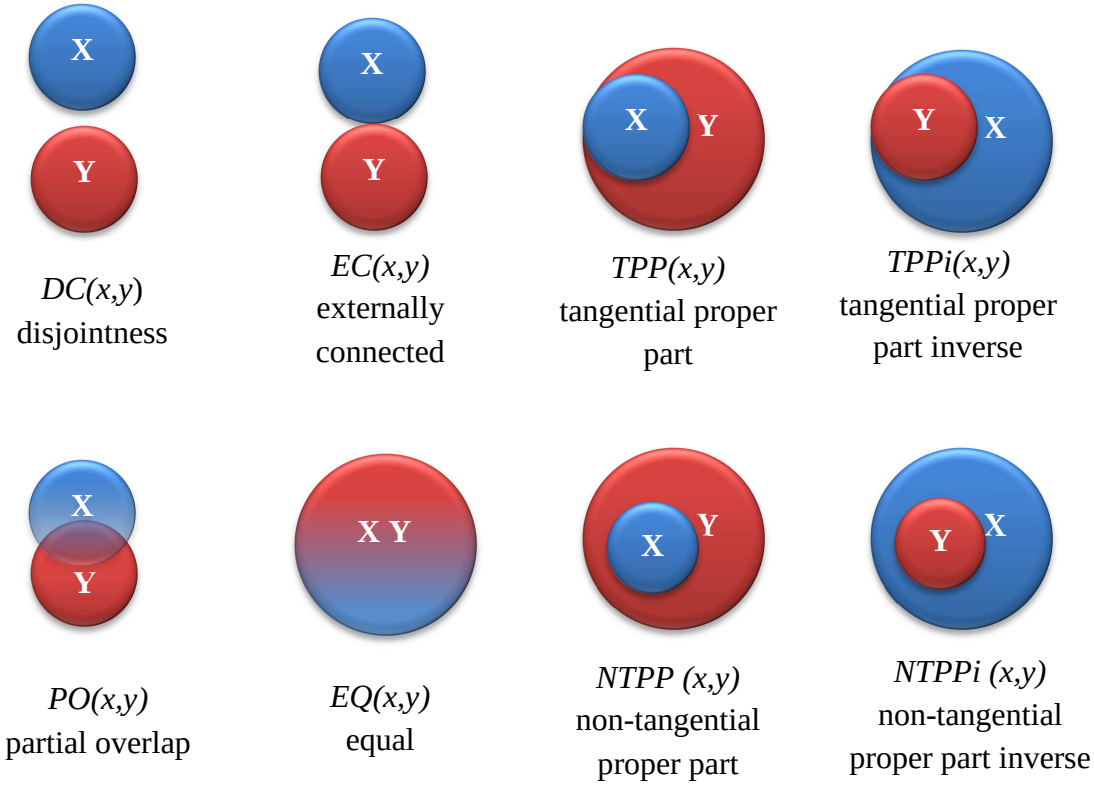

Figure 3.12 RCC8 Base Relations

B_{RCC8} denotes the eight base relations, graphically illustrated in Figure 3.12. For instance, the *DC(x, y)* relation means that region *x* and *y* are disconnected and they do not share a common point, while the meaning of *EC(x,y)* is that regions *x* and *y* only share borders. Besides, the RCC8 base relations have the JEPD property, namely they are jointly exhaustive and pairwise disjoints [32]. In other words, between any two regions *x,y* exactly one of the base relations holds, e.g. either only *DC(x,y)* or only *EC(x,y)* holds, but not both.

From a geographical point of view, in order to illustrate that *Germany* and *Poland* have a common border (see Figure 3.14), then this fact can be formalized by *EC(Germany, Poland)*. Another interesting base relation is the tangential proper part, meaning that the region *x* is contained in the region *y* and they share a part of the border from inside. In fact, it is possible to represent different geographical and spatial configurations of regions by using a set of disjunctions of base relations. This disjunctions have the form $r_1(x,y) \vee \ldots \vee r_n(x,y)$ or $\{ r_1, \ldots, r_n \}\{x,y\}$ [33], where *x* and *y* are constants and r_1 to r_n with $n \geq 1$ represent RCC8 base relations from Figure 3.12. In a nutshell, it is possible to express indefinite knowledge on the spatial relations of regions with the help of disjunctions of base relations. These disjunction statements are encapsulated in a network that consequently can be evaluated by various

constraint satisfaction algorithms. Referring again to the example of *Germany* and *Poland*, the following network is defined in Figure 3.13:

{ EC(Germany, Poland), DC(Germany, Bulgaria),
DC(Poland, Bulgaria), NTPP(Germany, Europe),
NTPP(Poland, Europe) , NTPP(Bulgaria, Europe) }

Figure 3.13 B_{RCC8} Network

Figure 3.14 Map of Europe [34]

The network in Figures 3.13 captures the geographical location of the countries, labeled in red in Figure 3.14, by representing the spatial constellation of three regions in terms of RCC8, stating that *Germany* and *Poland* are neighbors and *Bulgaria* does not share any border with any of the other two, but all of them are part of *Europe*. In general, it is also important to check whether this network is satisfiable. Satisfiability of networks can be tested with the help of path consistency algorithms, based on compositional tables. The composition table of B_{RCC8} is presented in Figure 3.16 [32]. The first row and first column of this table store the eight base relations and the other cells store respectively the composition of every pair of base relations. The *-operator specifies the universal relation, namely the disjunction of all base relations.

In fact, the table in Figure 3.16 encapsulates weak composition entries, namely minimal disjunctions of base relations [29]. A week composition is denoted with the symbol " ; " and it is an approximation of the composition, namely $r_1 ; r_2$ is implied by $r_1 \circ r_2$. Consulting the composition table, the weak composition for the pair *(EC, TPP)* can be provided, i.e. *{EC, PO, TPP, NTTP}*. Its description in Ax_{RCC8} (cf. Figure 3.17) is presented in Figure 3.15.

$$\forall x \ \forall y \ \forall z.(EC(x,y) \wedge TPP(y,z)) \rightarrow (EC(x,z) \vee PO(x,z) \vee TPP(x,z) \vee NTTP(x,z))$$

Figure 3.15. Weak Composition in Ax_{RCC8} Description

∘	***DC***	***EC***	***PO***	***TPP***	***NTPP***	***TPPi***	***NTPPi***	***EQ***
DC	*	*DC EC PO TPP NTPP*	*DC EC PO TPP NTPP*	*DC EC PO TPP NTPP*	*DC EC PO TP NTPP*	*DC*	*DC*	*DC*
EC	*DC EC PO TPPi NTPPi*	*DC EC PO TPP TPPi EQ*	*DC EC PO TPP NTPP*	*EC PO TPP NTPP*	*PO TPP NTPP*	*DC EC*	*DC*	*EC*
PO	*DC EC PO TPPi NTPPi*	*DC EC PO TPPi NTPPi*	*	*PO TPP NTPP*	*PO TPP NTPP*	*DC EC PO TPPi NTPPi*	*DC EC PO TPPi NTTPi*	*PO*
TPP	*DC*	*DC EC*	*DC EC PO TPP NTPP*	*TPP NTPP*	*NTPP*	*DC EC PO TPP TPPi EQ*	*DC EC PO TPPi NTPPi*	*TPP*
NTPP	*DC*	*DC*	*DC EC PO TPP NTPP*	*NTPP*	*NTPP*	*DC EC PO TPP NTPP*	*	*NTPP*
TPPi	*DC EC PO TPPi NTPPi*	*EC PO TPPi NTPPi*	*PO TPPi NTPPi*	*PO EQ TPP TPPi*	*PO TPP NTPP*	*TPPi NTPPi*	*NTPPi*	*TPPi*
NTPPi	*DC EC PO TPPi NTPPi*	*PO TPPi NTPPI*	*PO TPPi NTPPi*	*PO TPPi NTPPi*	*PO TPPi TPP NTPP NTPPi EQ*	*NTPPi*	*NTPPi*	*NTPPi*
EQ	*DC*	*EC*	*PO*	*TPP*	*NTPP*	*TPPi*	*NTPPi*	*EQ*

Figure 3.16 Composition Table of B_{RCC8}

Ax_{RCC8} is an axiom system schema providing axioms [29], [30], which directly state that the B_{RCC8} base relations are jointly exhaustive and pairwise disjoints. It is a weakened version of the original axioms of Randell, Cohn and Cui. In addition, this theory, shown in Figure 3.17 also provides axioms for weak composition and reflixivity of the equal base relation, i.e. *EQ*.

$$\{\forall x, y. \bigvee_{r \in \mathcal{B}_{RCC8}} r(x,y)\} \cup \qquad \text{(joint exhaustivity)}$$

$$\{\forall x, y. \bigwedge_{r_1, r_2 \in \mathcal{B}_{RCC8}, r_1 \neq r_2} r_1(x,y) \rightarrow \neg r_2(x,y)\} \cup \qquad \text{(pairwise disjointness)}$$

$$\{\forall x, y, z. r_1(x,y) \wedge r_2(y,z) \rightarrow r_3^1(x,z) \vee \cdots \vee r_r^k(x,z) \mid r; s = \{r_3^1, \ldots, r_3^k\}\} \cup$$

$$\text{(weak composition axioms)}$$

$$\{\forall x. \mathsf{eq}(x,x)\} \qquad \text{(reflexivity of eq)}$$

Figure 3.17 Ax_{RCC8} [30]

In fact, there are lower resolution logics than RCC_8, namely RCC_2, RCC_3, RCC_5 and according to [30], the Ax_{RCC8} is also applicable for the latter calculi. However, when considering RCC_2, the $\forall xEQ(x,x)$ from Figure 3.17 is replaced by $\forall xO(x,x)$, i.e. reflexivity of overlap. Since, the expressive power of RCC_2, RCC_3, RCC_5 is lower than that in comparison to RCC_8, the less expressive calculi are not recommended as powerful candidates to be combined strongly with DL-Lite for the creation of new spatial logics. That is the reason why the next chapter concentrates on the explanation of the DL-Lite(RCC8) formalism.

3.4.2 Lightweight DLs with RCC8

A solution proposal, dealing with the issue of reasoning over geo-thematic ontologies that involves accessing numerous databases, storing large volumes of spatial and topological data, is to combine Lightweight Description Logics with RCC8. The main issues when tailoring Lightweight Description Logics (e.g. DL-Lite) with Spatial Calculi (e.g. RCC) are to retain FOL rewritability with respect to both satisfiability and query answering, and to provide sufficient expressivity of the logics and the query language in order to facilitate an efficient and correct modeling and searching of GIS data. Several different perspectives are presented and investigated in [33, pp. 14-15] and [35, p. 8] in order to cope with the mentioned problems. In a nutshell, either a number of presupposed conditions on the ABox can be assumed or the combined logic can be weakened. The latter is achieved by either weakening the expressive power of the spatial calculus or the thematic part, or weakening the interaction of the combined components.

As a result, it is concluded that a reasonable compromise is to assume that the spatial data is consistent and complete, and the combination of DL-Lite and RCC is realized in a controlled way. In other words, a concrete solution, as mentioned at the beginning of this chapter, is to provide a weak coupling between DL-Lite and RCC8, predefining a spatial completeness condition for the ABox and using a restricted query language. A stronger coupling is also possible but only for the low-resolution region connection calculus RCC2, since otherwise FOL rewritability is not guaranteed any more [30, pp. 1, 6-10].

The combined logic DL-Lite(RCC8) allowing for FOL rewritability is illustrated in Figure 18, where the syntax and semantics of the logical constructors are presented and consequently explained. This approach both weakens the thematic part and restricts the interaction with the spatial component.

R	$\rightarrow$	$P \mid P^-$
U	$\rightarrow$	$loc \mid R \circ loc$
B	$\rightarrow$	$A \mid \exists R \mid \exists loc$
C	$\rightarrow$	$B \mid \neg B \mid \exists U_1,U_2.r$ for $r \in Rel_{RCC8}$ and not $(U_1=U_2=loc$ and $EQ \notin r)$
TBox		$B \sqsubseteq C$, *(funct loc)*, $R_1 \sqsubseteq R_2$
ABox		*A(a), R(a, b), loc(a, a*), r(a*, b*)*
T_ω	=	Ax_{RCC8}

Figure 3.18 Combined Logic DL-Lite(RCC8) [29, p. 4]

3. Analysis

A role *R* in DL-Lite(RCC8) is denoted in terms of a role symbol *P* or its inverses P^{-}. A path is defined either as a *loc* or as a composition of *R* and *loc*. The path legth is allowed to be at most 2. The left part *B* of an terminological axiom in the DL-Lite(RCC8) TBox can be represented by a concept symbol *A*, a limited existential quantification of a role symbol or attribute *loc*. The DL-Lite extension DL-Lite(RCC8) builds-up a weak coupling of the thematic and spatial domain, where apart from *B* and its negation $\neg B$, concepts of the form $\exists U_1,U_2.r$ are allowed to appear on the right-hand side *C* of the axioms in the TBox. However, only the concrete attribute *loc* (i.e. has location) may be functional. In addition, *r* stays for a set of all possible disjunctions of base relations from B_{RCC8}, i.e. *r* is a general RCC8 relation. Thus the set Rel_{RCC8} is the set of all (2^8-1) RCC8 relations, including the universal relation and excluding the empty relation. It is also assured that no empty concepts can appear by adding an extra restriction ($U_1=U_2=loc$ and $EQ \notin r$), because if for instance the right-hand side of an axiom is $\exists loc,loc.r$, then it denotes an empty concept in case that *EQ* is not part of the set *r*. Another alternative is to handle empty concept during the rewriting process. Furthermore, the ABox can contain assertions of the form *A(a), R(a, b), loc(a, a*), r(a*, b*),* where *a* and *b* are variables or constants and *a*,b** are also variables or constants, but intended to denote elements of Ax_{RCC8}.

As already mentioned, satisfiability in general is an important issue in logics. However, testing the satisfiability of arbitrary RCC8 constraint networks is not FOL rewritable [35, p. 7], making the process of checking a computationally intensive task. Consequently, it can be also concluded that coupling DL-Lite and RCC8 can also result in uncontrolled combinations, which are not FOL rewritable. For instance, by taking the query *ntpp(a*,b*),* it is searched in the database whether region *a** is a non-tangential proper part of region *b**. Nevertheless, the composition for the pair (*ntpp,ntpp*) from the composition table 3.16 yields that *ntpp* is a transitive relation. This relation could not be compiled into a finite FOL query, because all paths from *a** to *b** should be considered, i.e. $(ntpp(a^*,z_1^*) \wedge ntpp(z_1^*,z_2^*) \wedge ntpp(z_2^*,z_3^*) \wedge \ldots \wedge ntpp(z_n^*,b^*))$. Moreover, in real-world scenarios, it can be the case that spatial databases are incomplete. For instance, there could be a database entry that maps a parking in an airport terminal as a point, rather than as a polygon. As a result several of the base relations from B_{RCC8} may hold between the parking and the terminal, e.g. *PO(parking terminal), EC(parking, terminal), TPP(parking, terminal),* stating that it is not possible to decide whether the parking and the terminal partially overlap or they just touch each other either from outside or inside, etc. A solution not to face directly the issue of satisfiability and incompleteness is to assume that these issues are taken into account into an initial pre-processing step [35]. Hence, the notion of FOL rewritability for the combined logics using RCC8 as the spatial part is weakened by introducing a spatial completeness condition, i.e. a spatially complete ABox.

On the one hand, allowing FOL rewritability with respect to satisfiability testing is important in the context of combining lightweight DLs with RCC8. On the other hand FOL rewritability, considering query answering is also a fundamental factor. In a nutshell, the expressivity of the query language should be also taken into account. That is the reason why a querying language, called GCQ^+, is introduced in the next Chapter 3.4, since answering GCQ^+

queries within DL-Lite(RCC8)-ontologies with spatially complete ABox-es is FOL rewritable [29, p. 5].

3.4.3 Query Language GCQ+

GCQ^+ is a query language that is based on grounded conjunctive queries and it is appropriate for querying DL-Lite ontologies, since it both copes with the implausible consequences of the semantics of conjunctive queries and addresses the issue of computational unfeasibility of answering conjunctive queries with base relations in BRCC8 even if the ABox is interpreted to be complete [33, pp. 6,7,17]. Moreover, the GCQ^+ is explicitly build for DL-Lite(RCC8) ontologies and provides support for qualitative spatial query answering and possibilities for quantitative extensions. A GCQ^+ query atom has one of the forms, presented in Figure 3.19 [29, p. 5].

$$
\begin{aligned}
GCQ^+ \text{ atom} \quad &\rightarrow \quad C(x) \\
&\rightarrow \quad (\exists R_1 \ldots R_n.C)(x) \\
&\rightarrow \quad loc(x, y^*),\ y^* \in Rel_{RCC8} \\
&\rightarrow \quad r(x^*, y^*),\ r^* \in Rel_{RCC8} \text{ and } x^*, y^* \in Ax_{RCC8}
\end{aligned}
$$

Figure 3.19. GCQ^+ query atom

A GCQ^+ query atom may be a DL-Lite(RCC8) concept $C(x)$, where x is a variable or a constant, excluding the negation symbol. In addition, role symbols R or their inverses R^-, together with an existential quantifier $\exists$ and a concept C without the negation symbol can also build up a GCQ^+ query atom $(\exists R_1 \ldots R_n.C)(x)$. The last two representations include elements of Ax_{RCC8}, being part of the atoms $loc(x,y^*)$ and $r(x^*,y^*)$, defining the location y^* of x and the spatial orientation of regions x^* and y^*. A GCQ^+ query consists of conjunction of GCQ^+ query atoms. Furthermore, such a query can be compiled first into a UCQ with the help of a Perfect Rewriting Algorithm, explicitly tailored for dealing with geo-thematic scenarios, and as a result it can be transformed by an unfolding process into an SQL query, that can be fired to a spatial DBMS, assuming that the virtual ABox is spatially complete. These reformulation steps and corresponding techniques are presented consequently in following chapters.

3.5 Ontology Based Query Answering over Spatial Databases

Up to now the main issues and challenges referred to the OBDA technique, representation of spatial knowledge and reasoning about space have been addressed and discussed into details and numerous solutions and proposals have been provided. In a nutshell, the fundamental problems, tacked to so far, are:

- the trade-off between expressive power and computational complexity of ontology languages and numerous extensions

- large amount of data, stored at the source; data complexity
- linking data to ontologies
- the impedance mismatch problem

Answering more complex queries over ontologies is another fundamental requirement and challenging problem in Ontology Based Data Access. The complexity of the queries automatically implies that the query language should be more expressive than only specifying concepts and roles in DLs, namely it should be also able to express conjunctive queries and unions of conjunctive queries (UCQ).

3.5.1 Perfect Rewriting Algorithm

The DL-Lite$_A$ formalism is going to be used in order to initially illustrate the Perfect Rewriting Algorithm, i.e. *PerfectRef(Q,T)*, athough it is not expressive enough for dealing with spatial ontologies. This approach is reasonable, since the consequently presented version of the reformulation algorithms for DL-Lite(RCC8), i.e. *AdaptedPerfectRefQ,T)*, is based and explicitly uses the original Perfect Rewriting Algorithm.

The Perfect Rewriting algorithm lies in the center of the Ontology Based Query Answering process. This algorithm inherits its name from the fact that the input query q over the ontology is reformulated with the help of the Positive Inclusions (PIs) from the TBox T. It can be proved that the negative inclusion axioms do not have to be considered for the rewriting. However, they have effects on the satisfiability test and can be neglected only, as fas as rewriting is discussed. The query q can be either a CQ or UCQ. After the rewriting processing, the TBox is not of interest anymore and the reformulated query q' is evaluated over the ABox $DB(\mathcal{A})$, as if the ABox is a relational database. This process is graphically illustrated in Figure 3.20.

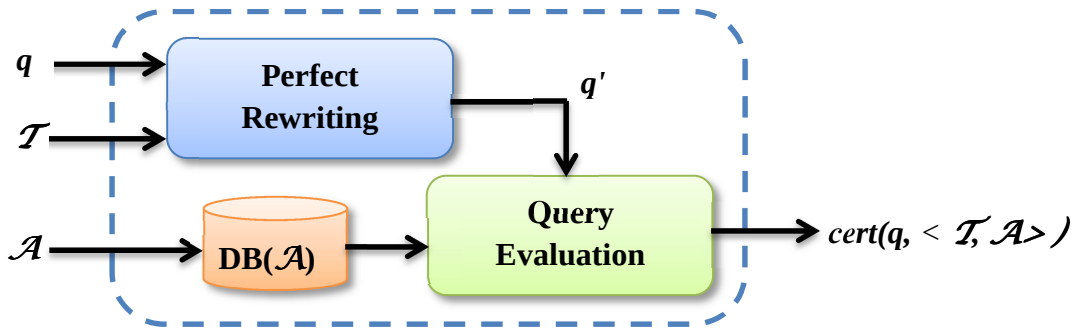

Figure 3.20. Ontology Based Query Answering [36, p. 93]

The answer to the initial query q over the ontology $O=<\mathcal{T}, \mathcal{A}>$ is the output of the illustrated process, namely the certain answer set $cert(q, <\mathcal{T}, \mathcal{A}>)$, being a tuple of constants of $\mathcal{A}$, which belong to q^I (answers to q over I) for every model I of the ontology O. Consequently,

analyzing Figure 3.20 and the latter definition of certain answer, it implies that $q^I = q^{DB(\mathcal{A})} = cert(q, <\mathcal{T}, \mathcal{A}>)$. Besides, the reformulated query *q'*, being an output of the block Perfect Rewriting, is in fact a UCQ and its size is independent of the size of the extensional level of the ontology, but it is exponential in the size of the TBox in the worst case. Nevertheless, the good news is that the query answering algorithm has a data complexity AC^0, since evaluating *q'* has a data complexity not worse than the traditional query evaluation in relational databases. The Perfect Rewriting Algorithm for computing the perfect reformulation of a conjunctive query with respect to a TBox in DL-Lite$_A$ Logics is presented in Figure 3.21.

input : a CQ q, DL-Lite$_A$ TBox $\mathcal{T}$
output : a UCQ *pr*

```
1  pr := q;
2  repeat
3      pr' := pr;
4      foreach CQ q' ∈ pr' do
5          foreach atom g in q' do                                  ┐
6              foreach PI α in T do                                 │
7                  if α is applicable to g then                     │
8                      pr := pr ∪ {q'[g/gr(g, α)]};                 ├ 1st part
9                  end                                              │
10             end                                                  │
11         end                                                      ┘
12         foreach pair of atoms g1, g2 in q' do                    ┐
13             if g1 and g2 unify then                              │
14                 pr := pr ∪ {anon(reduce(q', g1, g2))};           ├ 2nd part
15             end                                                  │
16         end                                                      ┘
17     end
18 until pr'=pr;
19 return pr;
```

Figure 3.21 PerfectRef Algorithm [7, p. 308]

Analyzing Figure 3.21, it is important to point out that the output of the algorithm is a set of CQs, generated on the basis of the input CQ. In fact, the input q is always the first element of the output UCQ *{pr}*, because of the assignment operation in line 1. After that a *do…while* loop starts, i.e. symbolically shown as *repeat…until*, which terminates when no more reformulations are possible any more. This condition is illustrated in line 18, where the set of reformulated CQs from the current and previous iterations is compared. The algorithm PercetRef always terminates [7, p. 309], since the length of the input conjunctive query and the maximum number of atoms in the body of the CQ that is generated, are equal. Moreover, the total number of distinct generated atoms is polynomial of the size of the input query. Line 7 checks whether a Positive Inclusion axiom α from the terminology $\mathcal{T}$ is applicable to an atom *g* from the conjunctive query *q'*. This is the case, when:

- the atom *g* is a an atom of the form *C(x)* and *C* is contained in the right-hand side of α;

- the atom *g* is an atom of the form. *R(x,y)*, and right hand-side of α is either *R* or R^-;
- the atom *g* is an atom pf the form *R(x,y)*, and *x* (respectively *y*) is a non-distinguished non-shared variable and the right hand side of α is $\exists R^-$ (respectively $\exists R$).

The result of applying the PI α to *g* is then represented as *gr(g, α)* in line 8. This result substitutes the current atom *g* and thus a new query is consequently added to the set of conjunctive queries *pr*. A detailed table, summarizing all cases when a given PI *a* is applicable to a query atom *g* and presenting the corresponding result *gr(g, α),* is shown in Figure 3.22.

atom g	PI α	gr(g, α)
$C(x)$	$C_1 \sqsubseteq C$	$C_1(x)$
$C(x)$	$\exists \boldsymbol{R} \sqsubseteq C$	$R(x, \_)$
$C(x)$	$\exists \boldsymbol{R^-} \sqsubseteq C$	$R(\_, x)$
$R(x, y)$	$R_1 \sqsubseteq R$ or $R_1^- \sqsubseteq R^-$	$R_1(x, y)$
$R(x, y)$	$R_1 \sqsubseteq R^-$ or $R_1^- \sqsubseteq R$	$R_1(y, x)$
$R(x, \_)$	$C \sqsubseteq \exists R$	$C(x)$
$R(x, \_)$	$\exists \boldsymbol{R_1} \sqsubseteq \exists R$	$R_1(x, \_)$
$R(x, \_)$	$\exists \boldsymbol{R_1^-} \sqsubseteq \exists R$	$R_1(\_, x)$
$R(\_, x)$	$C \sqsubseteq \exists R^-$	$C(x)$
$R(\_, x)$	$\exists \boldsymbol{R_1} \sqsubseteq \exists R^-$	$R_1(x, \_)$
$R(\_, x)$	$\exists \boldsymbol{R_1^-} \sqsubseteq \exists R^-$	$R_1(\_, x)$

Figure 3.22. Applying PI to an atom [7, p. 307]

With the first part of the algorithm, the knowledge from $\mathcal{T}$ relevant for answering the initial query *q* is compiled into a new reformulated query *pr* (lines 5-11). The second step (lines 12-16) describes a process, in which if any two atoms from the reformulated query *q'* of step can be unified, the functions *reduce(q', g_1, g_2)* and *anon(q'')* are consequently executed. The function *reduce(q', g_1, g_2)* performs the actual unification of the atom g_1 and g_2 and returns a new query *q''*, that is the input for the function *anon(q'')*. The latter function realizes variables anonymisation by substituting all unbound variables in *q''* with "_", i.e. the symbol representing a non-distinguished non-shared variable. An important side effect of the function *reduce(q', g_1, g_2)* is that it may make bound variables in *q'* unbound in *q''* due to the most general unifier between g_1 and g_2 and thus consequently in the next iteration of the first part of the Perfect Rewriting algorithm PIs, which were not applicable to *q'* atoms may become applicable to *q''* atoms.

The PerfectRef algorithm from Figure 3.21 is illustrated by practical examples, adopted from [7, pp. 308-310]. Figure 3.23 shows a DL-Lite TBox $\mathcal{T}$, where the atomic concepts *Teacher* and *Pupil* and atomic roles *TEACHES-TO* and *HAS-TUTOR* are defined. In addition, according to the latter TBox $\mathcal{T}$ no *Pupil* is also a *Teacher* and *Teachers* teach the *Pupils*, who have a tutor, being a *Teacher*. The functional role *HAS-TUTOR* represents a restriction that everyone has at most one tutor. A conjunctive query *q(x)* over $\mathcal{T}$ asks for *Teachers*, who teach to *Pupils*, having a tutor.

Teacher	⊑	¬*Pupil*	∃*HAS-TUTOR*⁻	⊑	*Teacher*
Teacher	⊑	∃*TEACHES-TO*	∃*TEACHES-TO*⁻	⊑	*Pupil*
Pupil	⊑	∃*HAS-TUTOR*	*(funct HAS-TUTOR)*		

CQ: q(x) ← TEACHES-TO(x, y), HAS-TUTOR(y, _)

Figure 3.23. School TBox $\mathcal{T}$ and a CQ over $\mathcal{T}$

By applying the algorithms of Figure 3.21 to the ontology $\mathcal{T}$ and the query *q(x)*, during the initial execution of the first part of the algorithm the positive inclusion *Pupil* ⊑ ∃*HAS-TUTOR* is applicable to the atom *HAS-TUTOR(y, _)*. As a result, the new query *q(x)* ← *TEACHES-TO(x, y), Pupil(y)* is added to the set of CQs *pr*. At the end of the next execution the query *q(x)*←*TEACHES-TO(x, y), TEACHES-TO(_, y)* is generated, which atoms unify and thus producing in the second part of the algorithm (line 12-16 from Figure 3.21) the new query *q(x)*←*TEACHES-TO(x, _)*. Comparing the last two queries, it can be observed that the bound variable *y* from the former query is unbound in the latter query, substituted by the symbol "_". The further executions of the Perfect Rewriting algorithm yield the queries *q(x)*←*Teacher(x)* and *q(x)*←*HAS-TUTOR(_, x)*. Finally, the initial query and the new generated queries are returned by the algorithm, thus reformulating the original query *q(x)* with respect to the TBox $\mathcal{T}$.

The initial purpose of query answering is to compute the answer of the original query *q* over the ontology $\mathcal{O} = < \mathcal{T}, \mathcal{A} >$ (cf. Figure 3.20). After creating a new reformulated UCQ out of the Perfect Rewriting algorithm, the next step is to evaluate the set of CQs *pr* over the ABox $\mathcal{A}$**,** i.e. to exploit the relational database *DB*($\mathcal{A}$). In order to achieve this, every CQ from *pr* should be transformed to an SQL query expressed over *DB*($\mathcal{A}$). In a nutshell, query evaluation and thus also query answering over satisfiable DL-Lite ontologies can be realized in an effective way by using the technology of RDBMS as defined in Figure 3.24 [37, p. 41].

cert(q, < $\mathcal{T}$, $\mathcal{A}$ >) = Eval(Unfold(PerfectRef(q, $\mathcal{T}$)), DB($\mathcal{A}$))

Figure 3.24. Computing Certain Answers to a Query
over DL-Lite$_A$ Ontology

Figure 3.24 illustrates the computation of certain answers to a CQ *q* over an ontology $O = < \mathcal{T}, \mathcal{A} >$ by first evaluating the *PerfectRef* algorithm from Figure 3.21, immediately followed by executing an *Unfold* function, which aim is to unfold the output UCQ query of the *PerfRef* and encode it in SQL. Finally, the *Eval* function evaluates the latter generated SQL query over a database DB.

If the TBox in Figure 3.23 is expanded by the role inclusion $HAS\text{-}TUTOR^{-} \sqsubseteq TEACHES\text{-}TO$ and the ABox in Figure 3.25 is also taken in order to build up an ontology, then reformulating the query $q(x) \leftarrow Pupil(x)$ over the TBox generates the UCQ=$\{ q(x) \leftarrow Pupil(x),\ q(x) \leftarrow TEACHES\text{-}TO(\_, x),\ q(x) \leftarrow HAS\text{-}TUTOR(x, \_) \}$. Consequently, the evaluation of the query q over the ontology yields the certain answer $q^{DB(A)} = cert(q, <\mathcal{T}, \mathcal{A}>) = \{Alex, Julia\}$. It is important to realize that without using the Perfect Rewriting algorithm the answer to the query $q(x) \leftarrow Pupil(x)$ would have been the empty set, since there is no instance of *Pupil* in the sample ABox in Figure 3.25. Moreover, without expanding the TBox, as previously described, the answer to the query would have been only *{Julia}*.

HAS-TUTOR(Alex, Mr. Schmidt) *TEACHES-TO(Mr. Schmidt, Julia)*

Figure 3.25 School ABox $\mathcal{A}$ and a CQ over $\mathcal{A}$

3.5.2 A Bottom-up Approach

According to [7], the easiest way to reason over ontologies with mappings is to make use of the mappings, thus generating the actual ABox out of the data source. Consequently, by using a query answering algorithm, described in Chapter 3.5.1, reasoning over the materialized ABox and original TBox can be performed. This technique is known in the literature as a naïve or bottom-up approach. The main drawback of this method is the fact that the actual ABox is produced from the data at the source and in this way the information is duplicated. Moreover if the data is very big, as in spatial databases, then this disadvantage becomes more problematic. In order to avoid this negative effect of duplicating the data, another approach is proposed in [7], [5], where the ABox is not explicitly built and it is kept virtual. This approach is known as the top-down approach.

In order to explain the two approaches in details, the terms virtual and materialized ABox should be clarified. As described in Chapter 3.2 an ontology with mappings consists of a TBox T, relational database D and mapping assertions M, namely O = < T, D, M >. Moreover, the ontology with mappings and its split version are equivalent. On the other hand, a virtual ABox in DL-Lite$_A$ or DL-Lite$_{A,id}$ consists of a set of membership assertions, which are computed on the basis of the mapping assertions and the data from the database. A formal description of a membership assertion and a virtual ABox [7] are presented respectively in Figures 3.26 and 3.27.

$$A(m,D) = \{m_{right}[x/v] \mid v \in ans(m_{left}, D)\}$$

Figure 3.26. Definition of a Membership Assertion

3. Analysis

A membership assertion *A(m,D),* generated by a (split) mapping assertion *m* (cf. Figure 3.6) from database *D***,** is in fact an atom of the right part of a corresponding mapping assertion m_{right}, where the variable/s *x* is/are substituted by the answer of evaluating the left part of the mapping assertion m_{left}, i.e. the SQL query, from the database *D*. For instance, if the table *University* from Figure 3.3 contains the entries *{(31897, Web Engineering), (23456, Mathematics)}* and the split mapping assertions M_{11} and M_{12} from Figure 3.6 are taken into account, then the membership assertions are defined as follows:

$$A(m_{11},D)=\{\ Student(\ std(31897))\ ,\ Student(\ std(31897))\}$$

$$A(m_{12},D)=\{\ ATTENDS(\ std(31897),\ lect(Web\ Engineering)\ )\ ,\ ATTENDS(\ std(31897),\ lect(Mathematics)\ )\ \}$$

Figure 3.27. Example of Membership Assertions

Consequently, if all membership assertions are obtained, as defined in Figure 3.27, the entire virtual ABox can be generated.

$$A(M,D)=\{\ \bigcup A(m,D)\ |\ m \in M\}$$

Figure 3.28. Definition of a Virtual ABox

In other words, by computing the *A(M,D)*, the virtual ABox is materialized. This is in fact the first step of the bottom-up approach for query answering over ontologies with mappings. The second step is to perform the query answering algorithm to the ontology $\boldsymbol{O} = < \boldsymbol{T}, \boldsymbol{A(M,D)} >$, that is presented in Chapter 3.5.1, where *A(M,D)* is a materialized ABox. In a nutshell, using rewriting and the materialization means that the construction of *A(M,D),* then the creation of the the herbrand model *DB(A(M,D))* and finally the evaluation of the rewritten query *q'* on *DB(A(M,D))* should be performed.

As already mentioned at the beginning of chapter the bottom-up approach has several disadvantages:

- Materialization and storage of the entire virtual ABox, i.e. computing the *A(M,D),* are required. Moreover, the virtual ABox is generally polynomial in the size of the relational database, meaning a generation of huge overhead is produced by duplication of information.

The first disadvantage has also an immediate negative effect on the data complexity of the resulting algorithm, since it is not anymore AC^0 or LogSpace in the size of the database, but it is PTime in the size of the database, since materialization is a problem in PTIME complexity

- From usability and practical point of view, a fundamental drawback is the fact that complex data refreshment procedures and mechanism should be invented in order to keep both the ontology and the database synchronized, since the data sources are independent from the ontology (cf. Figure 3.2).

3.5.3 A Top-down Approach

In order to overcome the issues and drawback, which the bottom-up approach has, a different top-down technique is proposed for query answering over ontologies with mappings [7, pp. 338-341]. The main distinguishing feature of the top-down approach in comparison to the first approach is that the materializing of the virtual ABox is avoided by using an additional unfolding algorithm. It makes immediate advantage of the mapping specifications and consequently generating a SQL query, that is issued over a RDBMS and its result set coincide with the results of the initial query over the ontology. Thus the data complexity of the entire algorithm is in AC^0 and no additional data refreshment procedures have to be implemented to keep the data in the database and the ABox synchronized. However, the mappings should be always updated in case structural changes in the database are carried out. The top-down approach consists of four important steps, graphically illustrated in Figure 3.29.

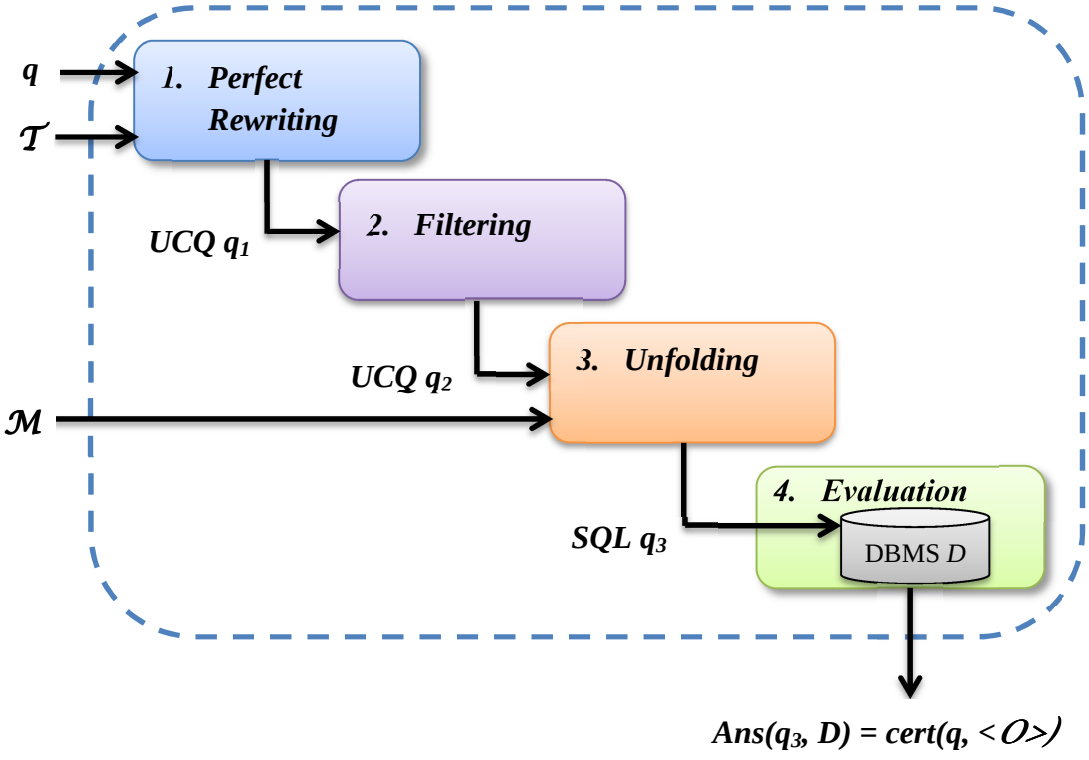

Figure 3.29. A Top-down Approach for Query Answering over Ontologies with Mappings

3. Analysis

Assuming that $O=<\mathcal{T},\mathcal{M},\mathcal{D}>$ is a DL-Lite$_A$ ontology with mappings, where $\mathcal{T}$ is the TBox, $\mathcal{M}$ represents the mapping assertion (i.e. split mappings) and $\mathcal{D}$ is the database. The first step of the approach is graphically illustrated in Figure 3.29 by a blue block. This block executes the Perfect Rewriting Algorithm as described in Figure 3.21. The output of the Reformulation step is a UCQ q_1 with the property that the "*certain answers to q with respect to O coincide with the set of tuples computed by evaluating q_1 over DB(A(M,D)), i.e. the database representing A(M,D)*" [7, p. 338]. The next step of the top-down approach performs a filtering process in order to get rid of every ill-typed conjunctive query, which contains join variables appearing in incompatible positions within the same query, thus producing a typing contradiction. The filtering step is beneficial for the next two steps in order to avoid producing wrong results in the query unfolding and query evaluation process over the source database *D*. The filtered query q_2 is the input for the Unfolding block, where with the help of the mappings $\mathcal{M}$, q_2 is transformed into an SQL query q_3, thus avoiding the materialization of the $A(\mathcal{M}, D)$ and evaluating the q_2 over $DB(A(\mathcal{M}, D))$. In fact, this is the main difference in comparison to the bottom-up approach. The Unfolding step is useful, since the result of executing the SQL query q_3 over the database *D*, i.e. the output of the SQL query Evaluation step, and the result of evaluating the q_2 over $DB(A(\mathcal{M}, D))$ coincide [7, p. 339] . Considering this fact and the previously described property of the output q_1 of the Perfect Rewriting step, it can be concluded that the certain answers to q with respect to *OM* coincide with the output of step number four. In other words, $cert(q,<\mathcal{O}>) = Ans(q_3, D)$.

3.6 Adapted Perfect Rewriting Algorithm

The Adapted Perfect Rewriting Algorithm, schematically illustrated in Figure 3.30, extends the original Perfect Rewriting Algorithm from Chapter 3.21 by also handling GCQ$^+$ atoms of the form $\exists U_1, U_2 .r$ for $r \in \text{Rel}_{\text{RCC8}}$ and it is based on the algorithm, proposed by [35, p. 11]. All modifications of the original Perfect Rewriting algorithm are marked within green rectangles in Figure 3.30. The input of the Adapted Algorithm is a hybrid query, the conjucts of which are either classical predicate logical atoms or GCQ$^+$ atoms. The aim is to transform this initial hybrid query into a UCQ and then by using the techniques, described in Chapter 3.5.3, to unfold the UCQ query to an SQL query that can be executed in a relational database, containing geo-thematic data. The other forms of GCQ$^+$ query atoms are treated as FOL-query atoms and they are processed by the original part of the Perfect Rewriting Algorithm (cf. Figure 3.30, lines 1-11, 33-40).

There are four relevant implications or four different cases for variations of GCQ$^+$ atoms, which are to be taken into account by the Adapted Perfect Rewriting Algorithm in Figure 3.30 [29, p. 5]:

1. If a GCQ+ atom of the form $\exists R_1 \circ loc, R_2 \circ loc.r_3\ (x)$ occurs during the rewriting process, then it can be substituted by the conjunct of two new atoms of the form $\exists R_1 \circ loc, loc.r_1\ (x)$ and $\exists loc, R_2 \circ loc.r_2\ (x)$ in a new CQ for all r_1, r_2, contained in Rel_{RCC8}

such that $r_1;r_2 \subseteq r_3$, namely where all possible compositions of the sets $(r_1 \circ r_2)$ from a full composition table are refinements (i.e. subsets) of r_3 (lines 13-16). A full composition table contains the compositions of all possible disjunctions of relations in Rel_{RCC8}, while the week composition table from Figure 3.16 embeds only the weak compositions of the 8 base relations in B_{RCC8}. The set of relations r_1 is the left argument (resp. a row element from the full composition table) and r_2 is the right argument (resp. a column element from the full composition table).

2. If a GCQ+ atom $\exists U_1, U_2.r_1\ (x)$ appears in the query and the TBox contains a terminological axiom of the form $B \sqsubseteq \exists U_1, U_2.r_2\ (x)$ and $r_1 \subseteq r_2$, then a new conjunctive query can be created with a query atom $B(x)$, substituting $\exists U_1, U_2.r_1\ (x)$ (lines 20-22)

3. The third case is similar to case 2, but it takes into account the inverses of the relations in Rel_{RCC8}. In fact, if a GCQ+ atom $\exists U_1, U_2.r_1\ (x)$ appears in the query and the TBox contains a terminological axiom of the form $B \sqsubseteq \exists U_2\ U_1.r_2\ (x)$ and $r_2^- \subseteq r_1$, then a new conjunctive query can be created with a query atom $B(x)$, substituting $\exists U_1, U_2.r_1(x)$ (lines 23-25).

4. If a GCQ+ atom of the form $\exists R_1 \circ loc, U_1.r\ (x)$ occurs as a conjunct in the query and the TBox contains a terminological axiom of the form $R_1 \sqsubseteq R_2$, then a new CQ can be created with a query atom $\exists R_2 \circ loc, U_1.r\ (x)$, substituting $\exists R_1 \circ loc, U_1.r_3\ (x)$ (lines 27-31).

input : a hybrid query Q, DL-Lite(RCC8) TBox $\mathcal{T}$
output : a UCQ *pr*

```
1  pr := Q;
2  repeat
3    pr' := pr;
4    foreach query q' ∈ pr' do
5        foreach atom g in q' do
6            if g is a FOL-atom then
7                foreach PI α in 𝒯 do
8                    if α is applicable to g then
9                        pr := pr ∪ {q'[g/gr(g, α)]};
10                   end
11               end
12           else
13               if g=∃R1 ∘ loc, R2 ∘ loc.r3(x) then
14                   foreach r1; r2 ⊆ r3 do
15                       X := q'[g/(∃R1 ∘ loc, loc .r1(x) ∧
                                    ∃loc, R2 ∘ loc .r2(x))];
16                       pr := pr ∪ {X}
17                   end
18               end
19               if g=∃U1, U2 .r1(x) then
20                   foreach B ⊑ ∃U1, U2 .r2(x) ∈ 𝒯 and r2 ⊆ r1 do
21                       pr := pr ∪ {q'[g/B(x)]}
22                   end
23                   foreach B ⊑ ∃U2, U1 .r2(x) ∈ 𝒯 for r2^-1 ⊆ r1 do
24                       pr := pr ∪ {q'[g/B(x)]}
25                   end
26               end
27               if g = ∃R1 ∘ loc, U1.r(x) (resp. ∃U1, R1 ∘ loc.r(x)) then
28                   foreach R2 ⊑ R1 ∈ 𝒯 or R2^-1 ⊑ R1^-1 ∈ 𝒯 do
29                       pr := pr ∪ { q'[g/(g[R1/R2])]}
30                   end
31               end
32           end
33       end
34       foreach pair of FOL-atoms g1, g2 in q' do
35           if g1 and g2 unify then
36               pr := pr ∪ {anon(reduce(q', g1, g2))};
37           end
38       end
39   end
40 until pr'=pr;
41 return drop(pr);
```

case 1 (lines 13–18)
case 2 (lines 19–22)
case 3 (lines 23–26)
case 4 (lines 27–31)

Figure 3.30. Adapted PerfectRef Algorithm

Finally, at the end of the Adapted Perfect Rewriting Algorithm the function *drop(pr)* (line 3.30) removes all GCQ+ queries, which contain atoms of the form $\exists U_1, U_2.r(x)$. As a result, the output of the algorithm is a classical UCQ, which can be consequently evaluated as a SQL query on the database $DB(\mathcal{A}(\mathcal{M},\mathcal{D}))$. The Figure 3.31 is adapted in order to illustrate the entire reformulation process of ontology based query answering over spatial databases, making use of a DL-Lite(RCC8) ontology, GCQ^+ query, data-to-ontology mappings and a PostgreSQL database, referred as a virtual and spatially complete ABox.

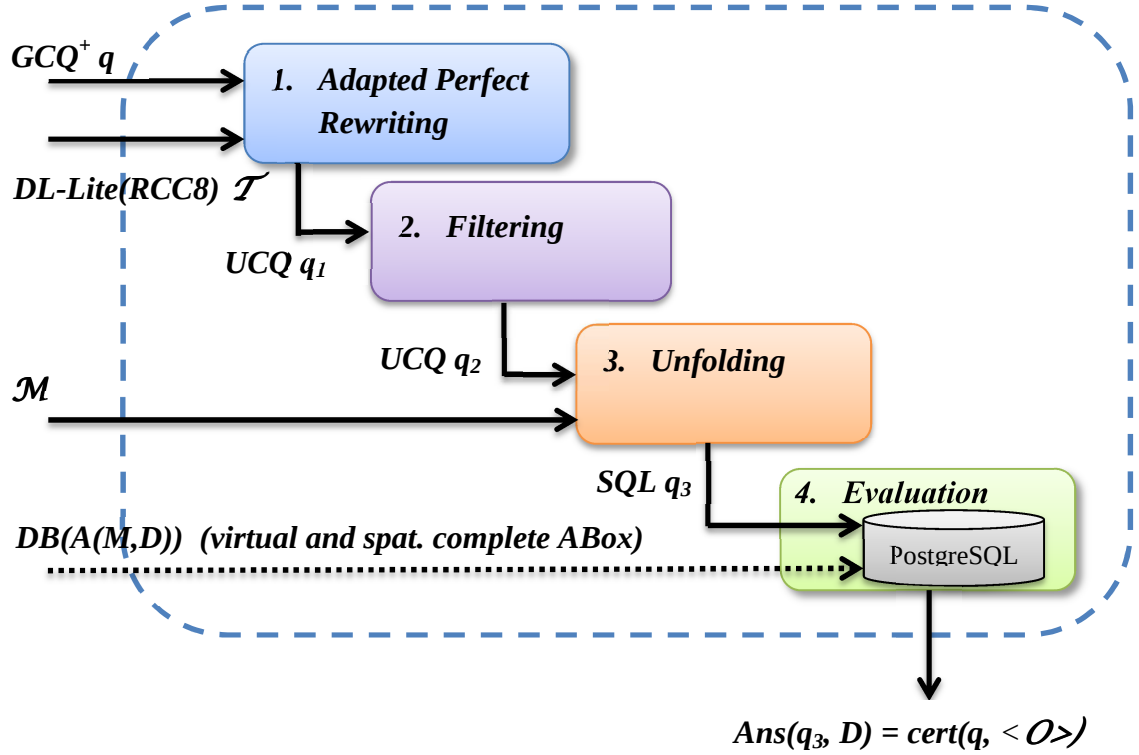

Figure 3.31. Ontology Based Query Answering over Spatial Databases

3. Analysis

3.7 Limitation Analysis and Final Requirements

In the latter sections, various techniques, general approaches and possible design solutions and algorithms for query answering over ontologies have been presented and theoretically analyzed without considering any external factors or limitations, which could consequently occur during the actual implementation of the desired software system. That is the reason why, this section reveals a list of all specifications, assumptions and obstacles that should be taken into account in the design and realization of the developed software application, implementing a Perfect Rewriting Algorithm for ontology based query answering over spatial databases.

Because of time constrains and the fact that the aim of this Master Thesis is not to develop a complete system for reasoning over spatial ontologies, the reasoning task such as ontology consistency checks, subsumtion between concepts, roles and attributes, and satisfiability tests are not taken into account. Moreover, it is decided not to provide the final system as a complete installation software package, but rather as a Java project that can be consequently imported in a universal tool for software development and executed in a debug mode. That is the reason why, only a simple graphical user interface will be designed in order to facilitate the user interaction, but also providing potential extensibility options. It is also assumed that the user inputs to the application are syntactically and semantically correct and the corresponding text files for specifying Tbox-es, mappings and search queries are well-formed and well-behaved.

The final requirements of this Master Thesis are presented in detail by dividing them into two main groups – theoretical requirements, specifying the theoretical approaches and algorithms to be implemented and program requirements, describing the system features and tasks to be realized by the developed software.

The final theoretical requirements are as follows:

- implement the Original Perfect Rewriting Algorithm for query answering over pure DL-Lite ontologies;
- extend the implementation of the Original Perfect Rewriting Algorithm to the Adapted Perfect Rewriting Algorithm over spatial ontologies, using the modified logic DL-Lite(RCC8) and the query Language GCQ^{+};
- incorporate the Adapted Perfect Rewriting Algorithm into a top-down approach for ontologies based Query Answering over spatial databases, thus avoiding the materialization of the virtual ABox by using an additional Unfolding step.

The main program requirement is to develop an ontology based Query Answering system that should provide:

- a framework for representing DL-Lite TBox, containing standard concepts, roles, role inverses, and inclusion assertions (i.e. PIs and NIs);

3. Analysis

- a framework for representing DL-Lite(RCC8) TBox, containing standard concepts, roles, role inverses, inclusion assertions (i.e. PIs and NIs) and concepts of the form $\exists U_1,U_2.r$, where $r \in Rel_{RCC8}$ and $U \rightarrow loc \mid R \circ loc$;
- a framework for representing conjunctive queries, containing query atoms of concepts, roles or GCQ^+ atoms of the form $\exists U_1,U_2.r(x)$;
- a framework for representing object-to-data mappings, containing mapping assertions of the form $M_{left} \sim> M_{right}$, such that the left part is and SQL query and the right part is a conjunction of atoms over the TBox;
- a Graphical User Interface for simple user interactions;
- a Parser for reading and interpreting the user inputs, i.e. a TBox, a conjunctive query and a set of mappings;
- a Reasoner for implementing the Original Perfect Rewriting Algorithm and the Adapted Perfect Rewriting Algorithm;
- a Reasoner for implementing the Query Reformulation process;
- a Resoner for evaluating the output of the Query Reformulation process over a PostgreSQL database.

The following two chapters of this Master Thesis present a detailed description of how the theoretical ideas are practically applied and how the listed requirements are realized and transferred into the design and implementation of a system for Query Answering over spatial ontologies.

4 Design

Following the overview of tools, technologies, query answering approaches, limitations, problem and system requirements analysis, this chapter reveals the actual design and architecture of the developed GIS application with DL-Lite(RCC8) ontologies.

4.1 Architecture Design

The system design of the geographical information system for ontology based query answering over spatial databases, abbreviated as OnQuAnSpatial, is graphically illustrated in Figure 4.1. The OnQuAnSpatial is a standalone Java application for the representation and reasoning over DL-Lite(RCC8) ontologies. The architecture of the program is divided into three main tiers, presented in Figure 4.1 as round-corner rectangular blocks, named as Frontend, Controller and Backend.

The Frontend specifies the top most level of the OnQuAnSpatial application, namely the the input text files and the graphical user interface. The main task of this tier is to realize a platform of interaction between the user and the software program and consequently display system outputs, hints and results of the ontology querying. The user defines a desired terminology in a text file, specifying the TBox with the DL-Lite(RCC8) syntax. Furthermore, it is possible to formulate a GCQ^+ query and mapping assertions in additional text files. Thanks to this layer, the user can interact with the application, which as a result passes the user requests down to the chain to the next layer, namely the Controller.

The second tier defines the business or domain logic of the application, where instances of concepts and roles in the ontology are retrieved from the data source tier, i.e. the Backend, processed and transferred to the Frontend, thus realizing a bidirectional flow of information among the different layers. Every text file is forwarded to the Parser component block, where numerous syntactical procedures validate the program inputs. If this process is successful, then the corresponding files are parsed accordingly and Java objects are created from the information they contain. This transformation step from a textual TBox, Query and Mappings to Java objects is facilitated by the blocks DL-Lite(RCC8) Interface, GCQ^+ Interface and Mapping Interface. As a result, the intensional part of the ontology and the GCQ^+ query are forwarded to the Adapted Perfect Rewriting block and the ontology-to-data-source mappings, realizing the formal linking of the stored persistent data and the ontology, are transferred to the Unfolding block. The latter two process blocks together with the Evaluation block from Figure 4.1 build up the most important component of the OnQuAnSpatial system, namely the Query Answering Reasoner that is explained in detail in Chapters 5.3 and 5.4. In addition, the component module Composition Table, storing the base relations from B_{RCC8} and the table of weak compositions (cf. Chapter 3.4.1, Figure 3.17), also takes part in the query reformulation

process. The connection between the Composition Table and Adapted Perfect Rewriting Algorithm blocks is represented by a dotted arrow, because the former is used only in case if the query *q* contains spatial atoms.

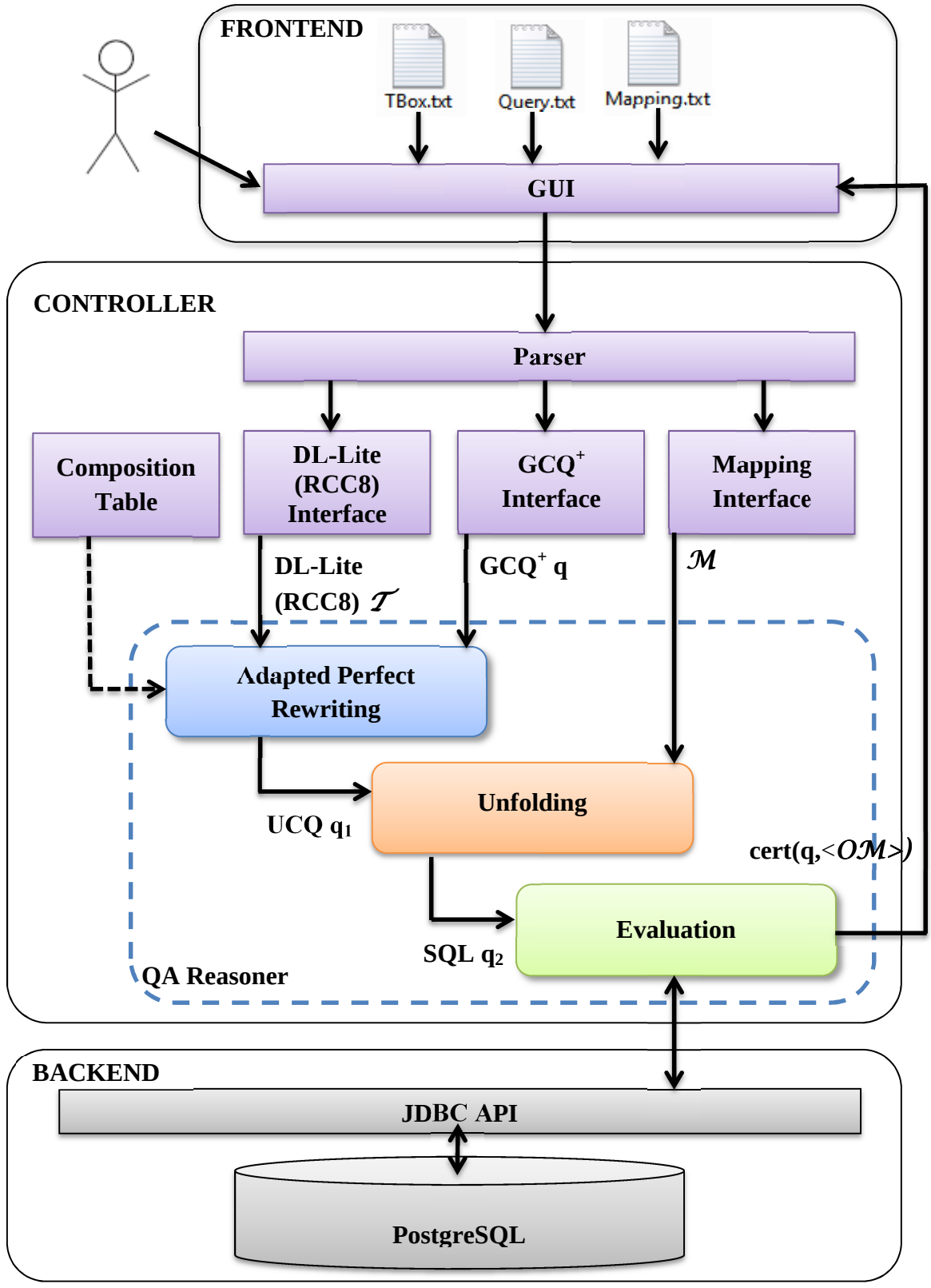

Figure 4.1. Architecture Design of the OnQuAnSpatial System with DL-Lite(RCC8)

The last tier of the OnQuAnSpatial system architecture from Figure 4.1 is the Backend. It is responsible for storing the extensional level of the ontologies and querying persistent data values within the DBMS PostgreSQL. The JDBC API enables the interaction of the application and the external data source, i.e. the database.

The main asset of the described architecture is that the input user data, application and database data are separated, thus providing possibilities for software scalability, modularization and reuse of components. Moreover, since the relationship between the instances of concepts and roles in the ontology and the data at the sources are realized with the help of mappings, it is not necessary to know how the data repository is organized or where the data is stored. In fact, the third layer, i.e. the background layer, can be theoretically modified or replaced by other databases without modification of the other two layers as long as the mappings are correspondingly modified. The drawbacks of this model refer to maintenance and complexity issues. Nevertheless, the 3-Tier architecture design, applied for the implementation of the OnQuAnSpatial system, proves to be very appropriate.

Last but not least, by comparing the typical design model of an information system, using ontologies from Figure 2.7 and the high-level architecture design of a DL knowledge representation system from Figure 2.6, it can be clearly concluded that the proposed architecture of the OnQuAnSpatial application incorporates both approaches. For instance, the knowledge base from Figure 2.6 matches the main components of the OnQuAnSpatial architecture, where the Description Logic block is corresponds to the DL-Lite(RCC8) component from Figure 4.1 and the TBox corresponds to the purple blocks from Figure 4.1, excluding the Mapping Interface module.

4.2 Software Prototype Design

Figure 4.2 reveals the package organization of the application and the corresponding classes and interfaces. The software packages *rocesses*, *dllitercc*, *mapping*, *queries*, *utilities* build the Controller as illustrated in Figure 4.1. The package *gui* and *repository* build the Frontend and the Backend layers of the application. Several stereotypes are applied to some of the packages in order to clarify the program design. For instance, the stereotype “DL-Lite(RCC8) Ontology” of the package *dllitercc* signals that this package consists of Java classes and interfaces, which represent an ontology in DL-Lite(RCC8). The main dependences among the packages are also shown as dotted import arrows, labeled with instructive names. The central point of the application is the package *processes*, where the actual reasoning procedures with respect to query answering are executed. The class Starter contains the main method of the program, which initializes the creation of Graphical User Interface menu frame, from where the user can consequently select text files, storing the information about the TBox, search query and mappings. After that, several processes can be triggered, depending on the user input.

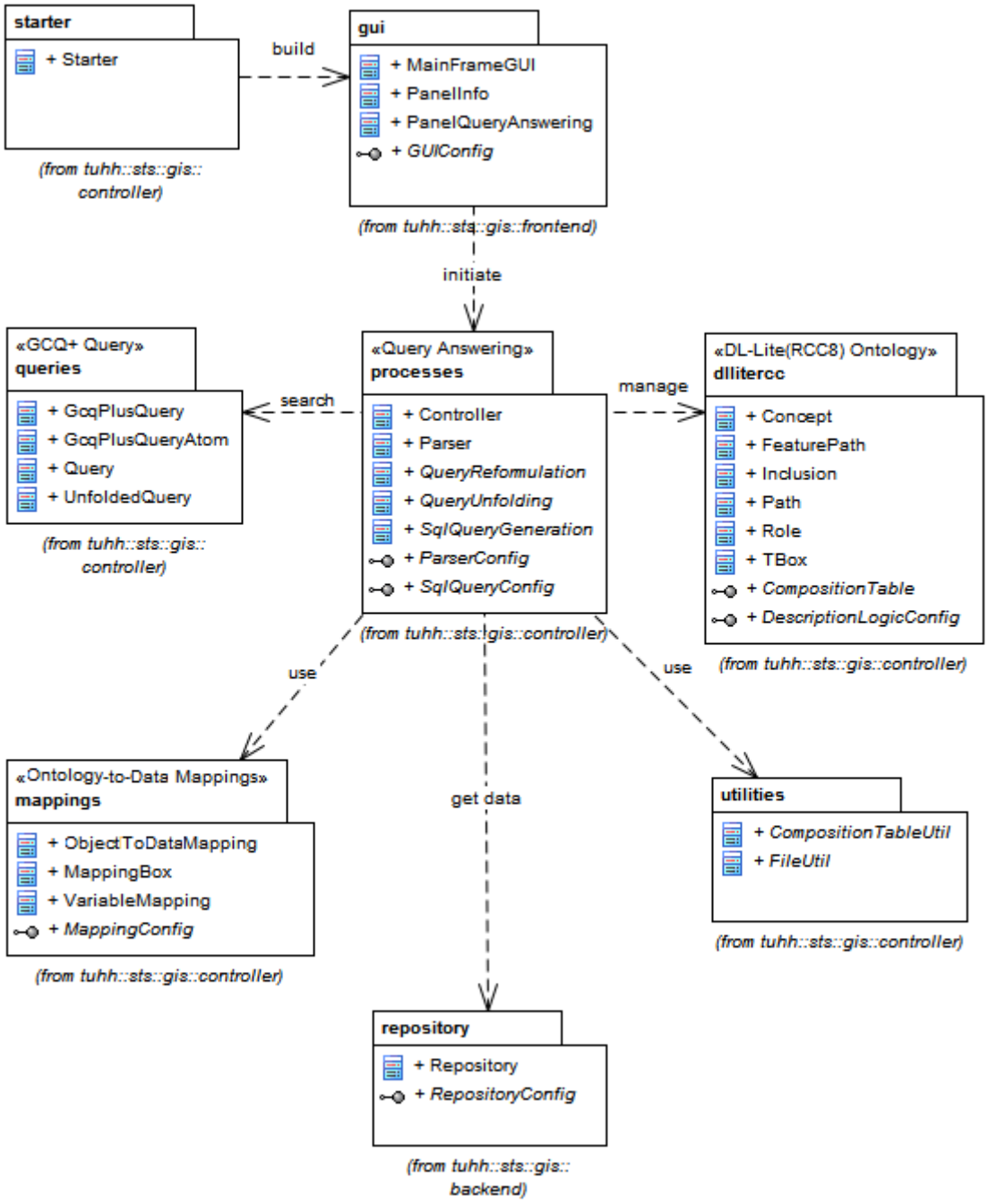

Figure 4.2. Software Package Diagram of the OnQuAnSpatial System

5 Realization

This chapter reveals how the main components of the OnQuAnSpatial system are realized by describing the most important software components, techniques and issues. Furthermore, the processes query reformulation and query unfolding are explained in more detail, followed by overviews of the application outputs and concluding discussions of the achieved results, based on the final system requirements outlined in Chapter 3.7.

5.1 Graphical User Interface

The OnQuAnSpatial software program is developed in Eclipse Java EE IDE for Web Developers, Version:Indigo Service Release 2, Build id:20120216-1857. The Java Runtime Environment (JRE) is JRE System Library [Java SE-1.7]. The used external library jar file is postgresql-9.1-902.jdbc4.jar in order to realize the connection between the database and the software application. The Graphical User Interface (GUI) is constructed with the help of the primary Java GUI widget toolkit Swing and the Abstract Window Toolkit (AWT) and it is illustrated in Figure 5.1.

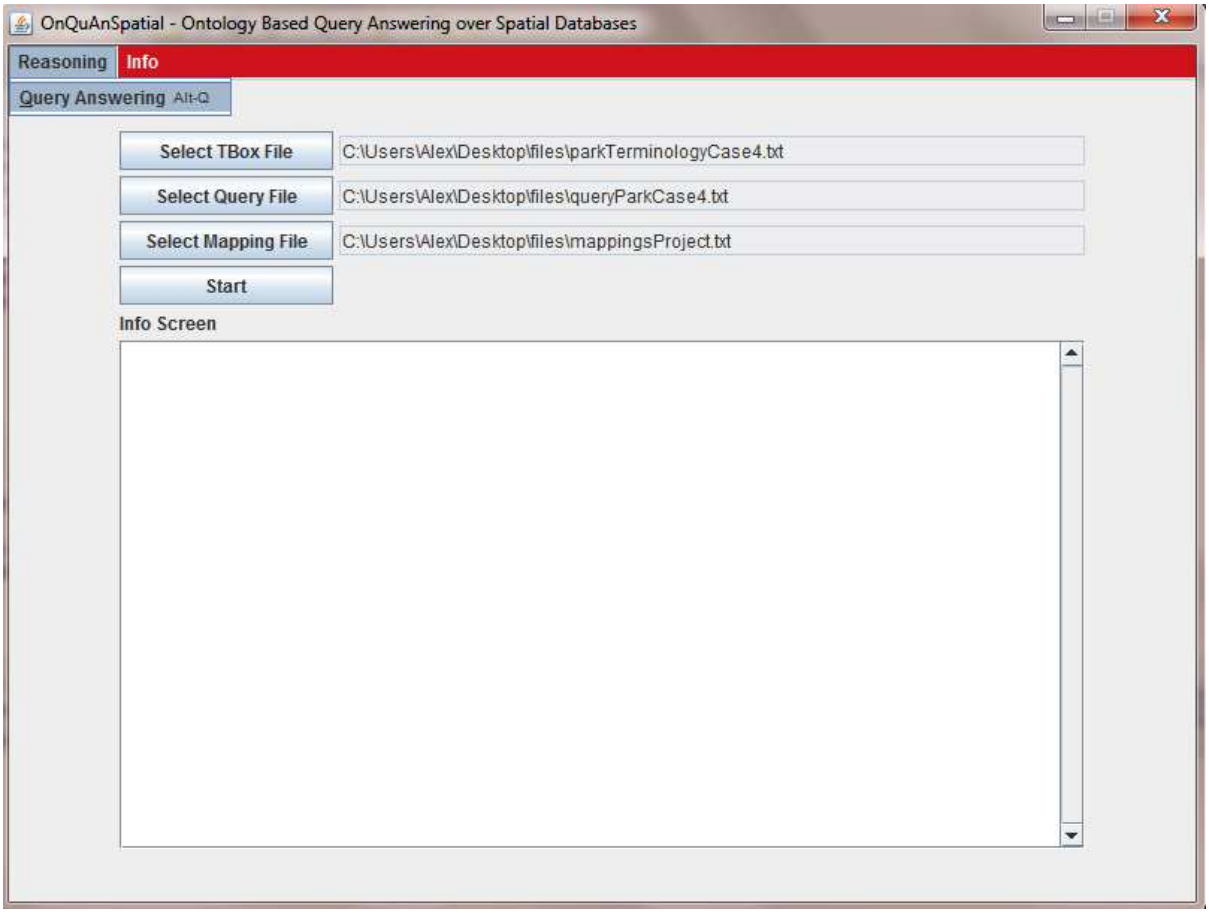

Figure 5.1. OnQuAnSpatial Prototype GUI

5. Realization

The user has the possibility to change several views, by navigating through the menu of the program, e.g. Reasoning, Info. In addition, it is possible to select different text files, containing the TBox, search query or the data-to-ontology mappings with the help of JFileChooser elements. This feature improves the software usability and flexibility, since the file locations should not be hardcoded in the program. When pressing the button “Start”, the query answering process begins. If no mapping file is selected, then only the Query Reformulation Part (cf Figure 4.1) will be executed without triggering the Unfolding and Evaluation processes. The Info Screen element from the GUI is meant to show program hints, error messages and results of querying DL-Lite(RCC8) ontologies.

The Unified Modeling Language (UML) diagram representing the GUI is illustrated in Figure 5.2. The associations between class *MainFrameGUI* and *PanelQueryAnswering* and respectively *PanelInfo* may be interpreted in fact as an aggregation, representing a “has a” relationship, i.e. the *MainFrameGUI* is a *JFrame* and it has two *JPanels*. The advantage of this GUI model design is that it can be easily extended by adding further *JPanel* classes, depending on the requirements of the software. Every class extending the *JPanel* from Figure 5.2 corresponds to an item from a drop-down menu (cf. Figure 5.1). The parameter lists from the method und constructor signatures are not displayed in order to achieve better readability of the UML class diagrams.

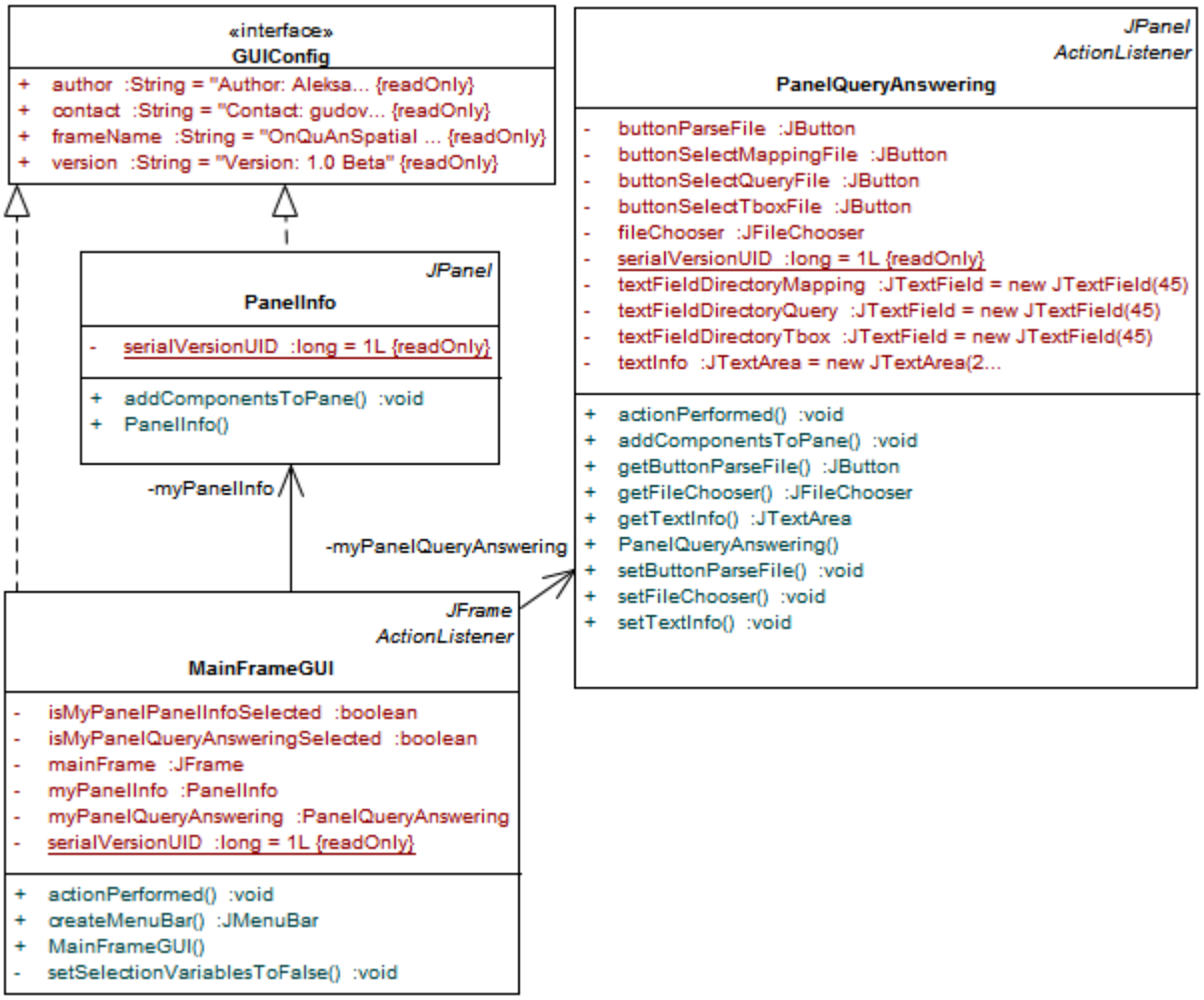

Figure 5.2. Class Diagrams of Package gui

5.2 TBox Implementation

The terminology of the ontology is parsed to Java objects from a text file. The OnQuAnSpatial application can handle both pure DL-Lite$_A$ or DL-Lite(RCC8) ontologies. An adapted example [7, p. 333] of a DL-Lite$_A$ TBox text file is shown in Figure 5.3, where information about employees and projects they work for is modeled. Managers and temporary employees are represented as employees (lines 11 and 12), who are persons. Both persons and projects have names and temporary employee has a date, indicating the expiration date of his contract (line 18). Moreover, everybody, having an attribute date, participates in the role *WORKS-FOR* (line 19) and every employee works for at least one project (lines 14 and 15). In conclusion, managers have permanent job positions (line 20). It is possible to include comments in the TBox text file by using the *;;;* symbol. All lines, beginning with the latter symbol are not considered by the Parser algorithm and only the logical operators and key words are interpreted. First of all the user should define the types of the terms he is using, namely concepts or roles. Functional assertions, expressing global functionality of a role or attributes are not understood by the application. In addition, attributes are defined as roles (e.g. *persName*, *until* on line 7 and 9). Inclusion assertions are recognized by the application by using the key-word *implies*.

```
1   ;;;TBox for projects
2   concept Manager
3   concept Employee
4   concept Person
5   concept TempEmp
6   concept Project
7   role persName
8   role projName
9   role until
10  role WORKS_FOR
11  Manager implies Employee
12  TempEmp implies Employee
13  Employee implies Person
14  Employee implies (some WORKS_FOR)
15  (some (inverse WORKS_FOR)) implies Project
16  Person implies persName
17  Project implies projName
18  TempEmp implies until
19  until implies (some WORKS_FOR)
20  Manager implies (not until)
```

Figure 5.3. Text File with a DL-Lite$_A$ TBox of a Project

The *Parser* of the OnQuAnSpatial application is case sensitive, meaning that it will interpret a concept *Manager* and *manager* as two different terms. In order to achieve better readability, it is recommended to first start defining all concepts by terms, starting with capital letters,

consequently listing all attributes (lines 7 to 9) and roles (line 10) as illustrated in Figure 5.3. When specifying the inclusion assertions, it is important to surround every concept or role term by parentheses always when a new logical operator is used, e.g. the existential quantification or inverse symbols (line 15), otherwise the parsing produces wrong results. All key words or logical operators are stored in the interface *ParserConfig* from the package *processes* (cf. Figure 4.2) and *DescriptionLogicConfig* from the package *dllitercc*. This approach guarantees consistent key-words and constants usage within all classes, implementing the latter interfaces.

Figure 5.4 illustrates a sample terminology [29, p. 5] within DL-Lite(RCC8) ontology that models parks (lines 2-4, 7-8), covering lakes (line 9) or playgrounds (line 10). The last two axioms from the TBox specify on the right hand side concepts of the form $\exists U_1, U_2 .r$, where U can be either *loc or* $R \circ$ *loc* and r is a general relation from Rel_{RCC8}. In case, a subset of relations (i.e. a disjunction of base relations) should be specified, then the base relations should be separated by comma in the form *(some HAS_LAKE *loc,loc.{tpp, nttp})*, stating that the lake can either touches the boundaries of the park from within or it can be an "island" in the park.

```
1   ;;;TBox for parks
2   concept Park
3   concept ParkWithLake
4   concept ParkForPlaying
5   role HAS_LAKE
6   role HAS_PLAYGR
7   ParkWithLake implies Park
8   ParkForPlaying implies Park
9   ParkWithLake implies (some HAS_LAKE *loc,loc.{tpp})
10  ParkForPlaying implies (some HAS_PLAYGR *loc,loc.{tpp})
```

Figure 5.4. Text File with a DL-Lite(RCC8) TBox of a Park

The OnQuAnSpatial system is able to read the two different text files and parse them accordingly into Java objects. As a result, instances of the classes *Concept*, *Roles* and *Inclusion* are created, which are building parts of a Java object of type *TBox*. These associations and the corresponding multiplicities are illustrated in Figure 5.5. The class *Concept* consists of two private attributes, namely a name and a term. A *Role* class differs than a *Concept* class by the fact that the former has two attributes. This differentiation on the number of terms is important for the Query Rewriting process, when concept and roles substitutes are searched in the TBox (cf. Figure 5.3) and when the *anon()* method is executed (cf. Figure 5.8), realizing variables anonymisation by substituting all unbound variables (i.e. terms) in the search query with "_" and thus consequently setting the non-distinguished non-shared variables. The class *Inclusion*, representing an axiom from the TBox, has a *left* and *right part* as attributes, as well as a type, being a "*positive*" or "*negative*" inclusion. This attributes are automatically set during the parsing process and besides that the OnQuAnSpatial system distinguishes the inclusions, containing concepts of the form $\mathbf{\exists U_1,U_2.r}$ on the right-hand side of the axiom. These types of spatial representation objects are

instances of the class *FeaturePath*, having the properties *constrain*, *paths*, *rccRelations* and *term*. The *paths* property represents a list of *Path* objects, i.e. DL-Lite(RCC8) concepts of the form **loc** or **R** o **loc**. *rccRelations* is also a list of objects, but of type *Role*. This list contains *Role* elements, which *name* attributes have any of the string values, saved in the *baseRCC* array from the interface *CompositionTable*.

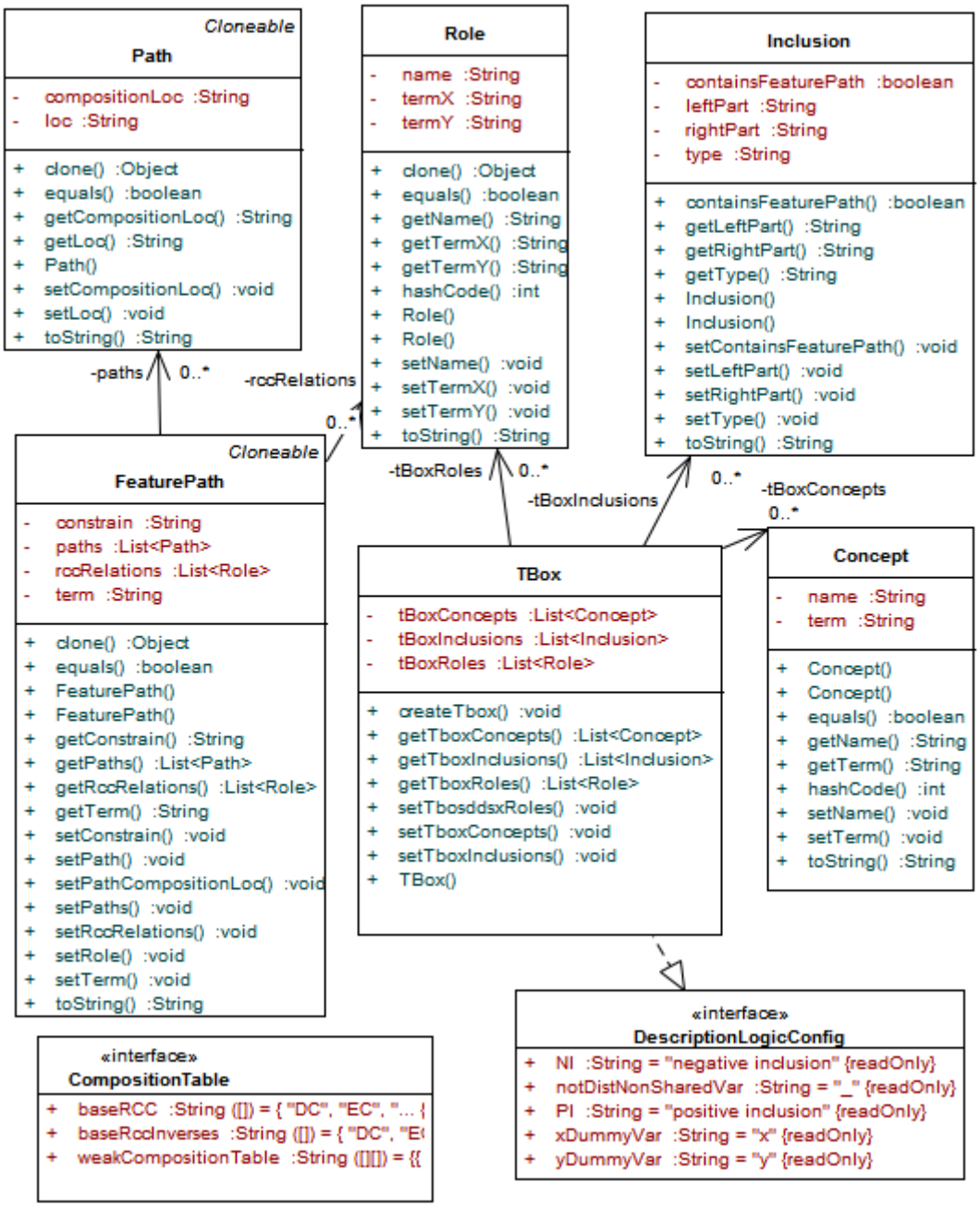

Figure 5.5. UML Class Diagrams of Package *dllitercc*

After initializations of the *TBox* object, the program flow continues with parsing the query text file. The advantage of using text files as an input for creating Java objects is that the

usability of this approach, because it is easy for the user to physically create these file. Moreover, they obey an intuitive DL syntax that is not significantly different from the syntax, presented in Figure 3.18. However, the main drawback is that a powerful checker and a validator should be implemented in order to identify all wrong user inputs. An alternative solution for parsing the user inputs is to make use of XML files, which can be verified against a predefined XML schema and automatically "*unmarshled*", i.e. translated, to Java objects. This can be achieved with the help of the Java Architecture for XML Binding API (JAXB). The described technique will on the one hand reduce the error rate of wrong user inputs and the complexity of the *Parser*, but on the other hand, it will also increase the difficulty for the user to specify these files, because additional XML knowledge should be provided.

5.3 Query Reformulation

The Query Reformulation process is part of the QA Reasoner, namely the blue block in Figure 4.1. Before starting the Perfect Rewriting Algorithm, the search query is retrieved from a query text file.

5.3.1 Input queries

Figure 5.6 illustrates two different types of search queries – a FOL conjunctive query without spatial atoms and a GCQ^+ query.

q(x,n) <- *WORKS_FOR(x,_) & persName(x,n)*

queryProject.txt

q(x) <- *Park(x) & some HAS_LAKE*loc,HAS_PLAYGR*loc.{ dc,ec,po,tpp,tppi,ntppi,eq}(x)*

queryPark.txt

Figure 5.6. Sample Query Files

The first query over the Project ontology from Figure 5.3 asks for all participants from the role WORK_FOR, who work for any project and their corresponding names. The second variable of the role WORK_FOR is unbound, that is the reason why it should be initially substituted by "_". The GCQ^+ query from the file queryPark.txt should search for all parks with lakes and playgrounds, such that the playground is not contained in the lake as an island, i.e. the playground is not in non-tangential proper part relation to the lake. The input files are translated to Java objects by the Parser and the GCQ^+ interface blocks from Figure 4.1. The corresponding classes for executing these tasks are listed in Figure 5.7.

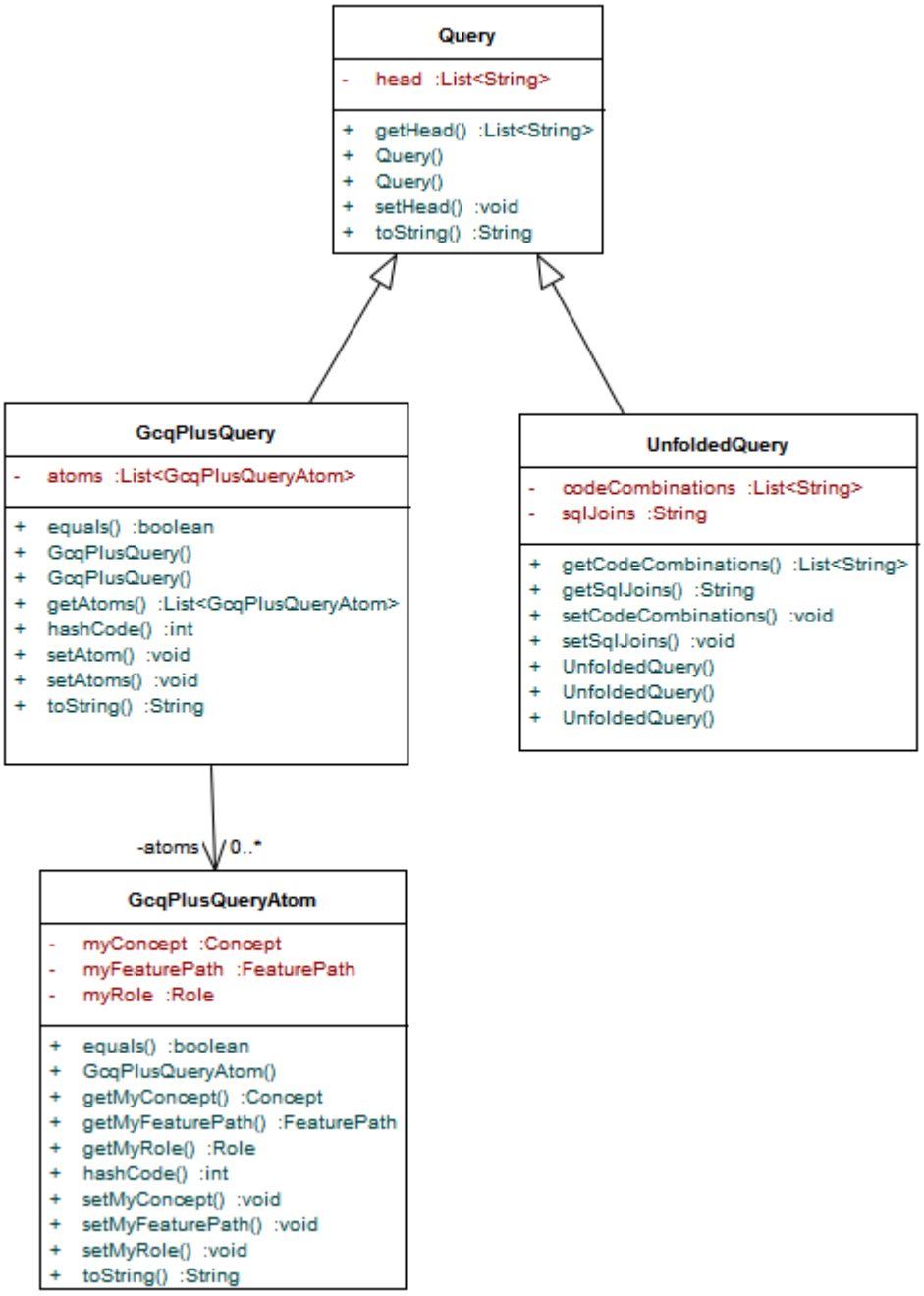

Figure 5.7. UML Class Diagrams of Package *queries*

The generalization relationship between the super class *Query* and the subclasses *GcqPlusQuery* and *UnfoldedQuery* is illustrated by arrows with hollow triangular endings. In other words, any instances of the subclasses are also instances of the superclass, i.e. a *GsqPlusQuery* is a *Query* and *UnfoldedQuery* is also a *Query*. This is a typical example of inheritance, since both child classes, which represent the query body or query tail, inherit all non-private class members from the parent class *Query*, representing the query head. Nevertheless, the private field head can be accessed indirectly by the inherited public methods *setHead*(…) and *getHead*(). The attribute *head* is important, since it has a fundamental influence in the *anon( )* method (cf. Figure 3.30, line 36) when defining which variables are

distinguished and thus also enabling indirectly the identification of all unbound variables. Moreover, it is also crucial when checking the variable consistency of the unfolded queries during the Query Unfolding step. Some of the constructors in Figure 5.7 are shown with the same signatures, but their parameter lists are not displayed, i.e. these are overloaded constructors. Furthermore, a *GsqPLusQuery* consists of a number of *GsqPlusQueryAtom*-s. In order to distinguish between a normal and a spatial query atom, it was decided that an instance of the class *GsqPlusQueryAtom* can either have a *Concept*, a *Role* or a *FeaturePath* istance. The *UnfoldedQuery* class is used in order to represent a query, generated during the Unfolding process, described in Chapter 5.4.

5.3.2 Implementation of Adapted Perfect Rewriting Algorithm

The implementation of the Adapted Perfect Rewriting Algorithms is based on the pseudo code, described in Chapter 3.6. Nevertheless, slight modifications and improvements are realized in order to optimize the reformulation process. The Adapted Perfect Rewriting Algorithm is implemented in the abstract class *QueryRepormulation* from the package *processes*. The corresponding UML class diagram is shown in Figure 5.8 and the entire source code is provided in the Appendix C.

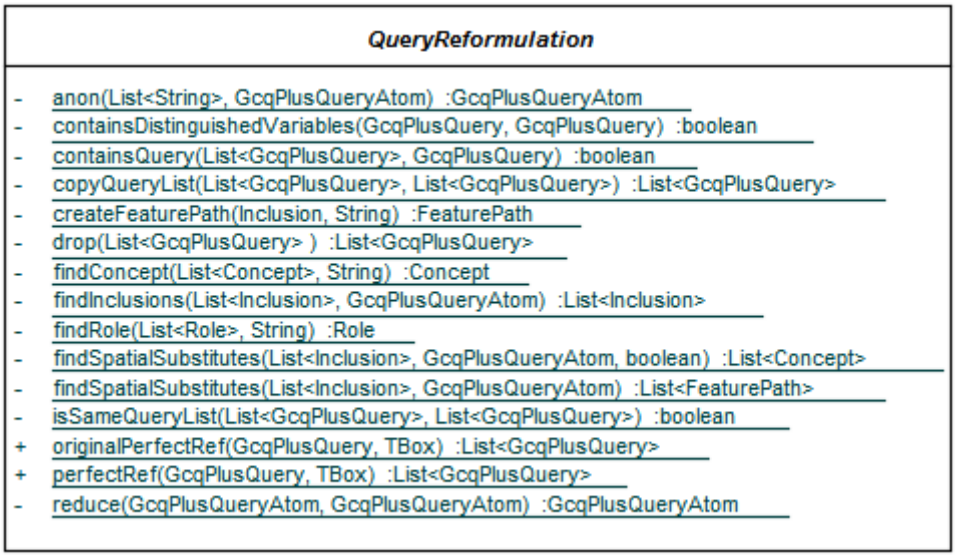

Figure 5.8. UML Class Diagram of *QueryReformulation*

The class *QueryReformulation* consists only of private methods, except for the method *perfectRef(…)* and *originalPerfectRef*(), which are public, since it is called from within the *Controller* class in order to trigger the rewriting process. All other methods are called internally within the method *perfectRef(…)* or *originalPerfectRef(…).* The signatures of these methods are listed in Figure 5.18, by also showing the types of the variables from the parameter list and the return variables. The operation *originalPerfectRef(…)* implements the original Perfect Rewriting Algorithm (cf. Figure 3.21), while *perfectRef(…)* implements

respectively the adapted version. The latter method has two input objects, namely a *query* of type *GcqPLusQuery*, as explained in Chapter 5.31, and an object *myTBox* of type *TBox*, as explained in Chapter 5.2. The method starts with variable initialization and a test procedure, going through all atoms of the input query in order to check whether an atom of the form $\exists R_1$ o *loc*, R_2 o $loc.r_3$ *(x)* occurs. If that is the case, then a full composition table is created and saved in a two dimensional array of strings. The generation of the composition table is a complicated process and it is described separately in the next chapter. After that a while-loop is started. It uses the method *isSameQueryList(queriesP, queriesPprime)* in order to compare whether both *query* lists are the same. If that is the case, then the while loop and respectively the reformulation algorithm terminate and the result is printed to the screen. The parameter *queriesP* contains the list of current queries and the *queriesPrime* contain the list, to which a new rewritten *query* is added. At the beginning of the while-loop the queries of the *queriesP* are copied to the *queriesPprime* by calling the method *copyQueryList(…).* After that for every non-spatial atom of the corresponding input query and for every PI from the object *myTBox*, it is checked in descending order whether a substitution is possible to a specific atom by calling the method *findInclusions(…),* which takes into account the possible cases from Figure 3.22, when and how to apply a PI to a query atom. Consequently, the output of the *findInclusions(…)* is additionally filtered in order to avoid adding the same query twice.

The next part of the adapted algorithm processes every spatial query atom of the form $\exists U_1$, $U_2.r_1$ *(x),* as described in Figure 3.30 (lines 12-32). For the case, that the atom of type *GcQPlusQueryAtom* has the form $\exists R_1$ o *loc*, R_2 o $loc.r_3$ *(x),* then it is substituted by the conjunct of two new atoms of the form $\exists R_1$ o *loc*, $loc.r_1$ *(x)* and $\exists loc$, R_2 o $loc.r_2$ *(x)* in a new *GcQPlusQuery* object for all possible sets r_1 and sets r_2 from the full composition table, being refinements (subsets) of the set r_3, i.e. $r_1;r_2 \subseteq r_3$**.** However, several performance tests have been carried out, concluding that this operation is computationally very expensive. This is in fact not surprising, since the full composition table has 255 rows and 255 columns, i.e. 2^8-1 RCC8 relations altogether, excluding the empty relation and including the universal relation. This results in 65025 possible combinations of pairs (r_1,r_2) or 65025 cells in the full composition table, excluding the vertical and horizontal headers. In terms of the software implementation of the algorithm, the proposed pseudo code in Figure 3.30, line (13-17) will generate up to maximum of 130050 new query atoms and respectively 65025 new queries for every input query atom in the form $\exists R_1$ o *loc*, R_2 o $loc.r_3$ *(x),* which is not acceptable from computational point of view. The former described performance tests check for every single set of disjunctions of base relations in RCC8 (i.e. r_3=255 possibilities), how many possible pairs satisfy the condition $r_1;r_2 \subseteq r_3$. All results are listed in Appendix C. However, Figure 5.9 presents the minimum, maximum and average number of pairs.

	possible pairs (r_1,r_2), such that $r_1;r_2 \subseteq r_3$	r_3
MIN	16384	*{(EQ)}*
MAX	65025	*{(EQ),(NTPPI),(TPPI),(NTPP), (TPP),(PO),(EC),(DC)}*
MEAN VALUE	~56918	-

Figure 5.9. Number of Possible (r_1, r_2) Pairs

The second column from the table in Figure 5.9 (possible pairs), also coincides with the number of added *GcQPlusQuery* objects to the initial spatial query in case the algorithm is not optimized. Furthermore, it is not surprising that in the case when r_3 represents the disjunctions *{ (EQ), (NTPPI), (TPPI), (NTPP), (TPP), (PO), (EC), (DC) }*, namely when r_3 is the universal relation, then the maximum number of compositions occurs. The reason of this phenomenon is the fact that in this case r_3 expresses the maximum indefinite knowledge on the spatial relations of regions.

The Adapted Perfect Rewriting algorithm, referring to the first case (c.f. Figure 3.30, lines 13-18), should be optimized with the extension that it does not search for all $r_1;r_2 \subseteq r_3$, but it seeks for all the maximal pairs r_1,r_2 such that $r_1;r_2 \subseteq r_3$ and does the reformulation process only w.r.t. these pairs. For instance, if on the one hand there is a pair r_1,r_2 *such that* $r_1;r_2 \subseteq r_3$ and on the other hand, a second pair exists r_4,r_5 such that $r_4;r_5 \subseteq r_3$ and moreover $r_4 \subseteq r_1$ and $r_5 \subseteq r_2$, then the pair r_4,r_5 is redundant, since the pair r_1,r_2 is the super set or the maximal pair withinn two pairs. The optimization of the Perfect Rewriting Algorithm is presented in Figure 5.10, by explicitly illustrating the improved modifications in a red rectangular block.

The proposed optimization improvement is implemented by calling the public method *getRowsColumnsFromStrongCompTable(…)* from the abstract class *CompositionTableUtil*, which belongs to the package *utilities.* After that the method *perfectRef* continues its execution by handling case 2 and 3 from the Adapted Perfect Algorithm. This is achieved by a double execution of the the method *findSpatialSubstitutes(…),* that returns a list of *Concept* objects. Case 4 is addressed in the private method *findSpatialSubstitutes(…),* returning a list of *FeaturePath* objects. Finally, the methods *reduce(…), anon(…)* and *drop(...)* are called in order to finalize the algorithm. The output of the method *perfectRef* from the class *QueryReformulation* is revealed in Chapter 5.3.4.

input : a hybrid CQ query Q, DL-Lite(RCC8) TBox $\mathcal{T}$
output : a UCQ *pr*

```
1  pr := Q;
2  repeat
3    pr' := pr;
4    foreach query q' ∈ pr' do
5        foreach atom g in q' do
6            if g is a FOL-atom then
7                foreach PI α in 𝒯 do
8                    if α is applicable to g then
9                        pr := pr ∪ {q'[g/gr(g, α)]};
10                   end
11               end
12           else
13               if g=∃R1 ∘ loc, R2 ∘ loc.r3(x) then
14                   foreach r1; r2 ⊆ r3 and r1; r2 is MAX do
15                       X := q'[g/(∃R1 ∘ loc, loc .r1(x) ∧
                                 ∃loc, R2 ∘ loc .r2(x))];
16                       pr := pr ∪ {X}
17                   end
18               end
19               if g=∃U1, U2 .r1(x) then
20                   foreach B ⊑ ∃U1, U2 .r2(x) ∈ 𝒯 and r2 ⊆ r1 do
21                       pr := pr ∪ {q'[g/B(x)]}
22                   end
23                   foreach B ⊑ ∃U2, U1 .r2(x) ∈ 𝒯 for r2^-1 ⊆ r1 do
24                       pr := pr ∪ {q'[g/B(x)]}
25                   end
26               end
27               if g = ∃R1 ∘ loc, U1.r(x) (resp. ∃U1, R1 ∘ loc.r(x)) then
28                   foreach R2 ⊑ R1 ∈ 𝒯 or R2^-1 ⊑ R1^-1 ∈ 𝒯 do
29                       pr := pr ∪ { q'[g/(g[R1/R2])]}
30                   end
31               end
32           end
33       end
34       foreach pair of FOL-atoms g1, g2 in q' do
35           if g1 and g2 unify then
36               pr := pr ∪ {anon(reduce(q', g1, g2))};
37           end
38       end
39   end
40 until pr'=pr;
41 return drop(pr);
```

case 1 (lines 13–18)
case 2 (lines 20–22)
case 3 (lines 23–25)
case 4 (lines 27–31)

Figure 5.10. Optimization of the Adapted PerfectRef Algorithm

5.3.3 Full Composition Table

The full composition table is also called strong composition table in the context of this Master Thesis in order to differentiate it from the table of weak composition in Figure 5. The creation of the former table is realized dynamically only if this is required by consequently calling the public methods *createRowColumnHeaderStrongCompTable( )* and *createStrongCompositionTable( )* from the abstract class *CompositionTableUtil*, presented in Figure 5.11.

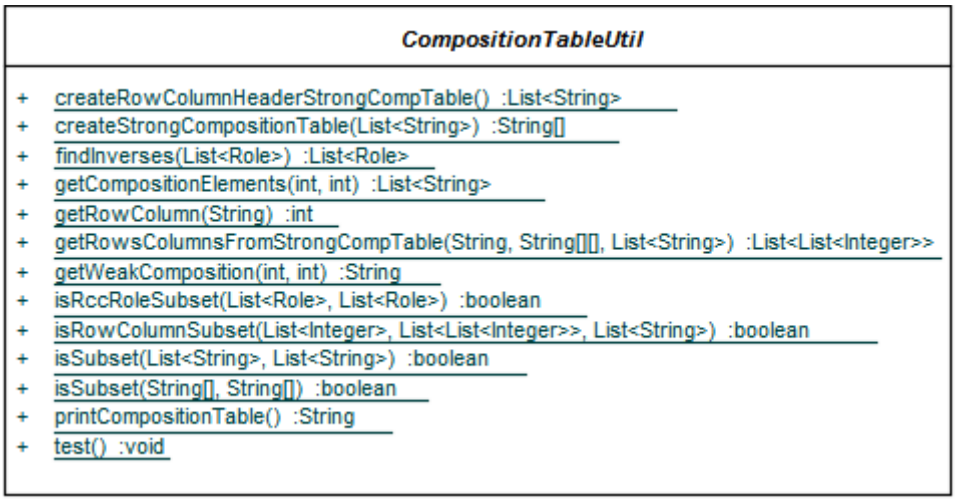

Figure 5.11. UML Class Diagram of *CompositionTableUtil*

The operation *createRowColumnHeaderStrongCompTable( )* returns a list of Strings, storing the vertical and the horizontal headers of the strong composition table. Both headers are identical, so that is the reason why the generation of only one of them is sufficient, which is assigned to the variable *rowColumnHeaderStrongCompTable*. An element of the list *rowColumnHeaderStrongCompTable* stores all possible disjunction combinations of the base relations in B_{RCC8}, namely *{(EQ),(NTPPI),(TPPI),(NTPP),(TPP),(PO),(EC),(DC)}*. In order to create the total 255 possible combinations, the method *createRowColumnHeaderStrongCompTable( )* uses a mathematical algorithm, inspired by the approach when building a simple truth table. Once the headers are generated, then any two vertical and horizontal cells are combined in order to get the corresponding compositions. For instance, if (r_{vert_2} o r_{horiz_3}) should be calculated, i.e. ({*EC*} o {*EC, DC*}), then the method *createStrongCompositionTable ( )* computes the intermediate result ({*EC ; EC*} or {*EC ; DC*}), which is directly reformulated in ({*DC, EC, PO, TPP, TPPI, EQ}* or *{DC, EC, PO, TPPI, NTPPI*}), by consulting the weak composition table from the interface *CompositionTable* (cf. Figure 5.12). As already mentioned, the strong composition table is needed when considering case 1 from the Adapted Perfect Algorithm in Figure 5.10.

The advantage of the described implementation for generating a full composition table is that the table should not be persistently stored and it is initialized only once if it is required. Moreover, in case that it is desired to use a composition table for RCC5, RCC3 or RCC2, then

only the corresponding base relations (line 13), their inverses (line 14) and their weak composition table (line 21) should be manually updated in the interface *CompositionTable* and the new full composition table can be generated automatically without any further modifications. A code snipped, illustrating the constants from the interface *CompositionTable* is presented in Figure 5.12.

```
 8  public interface CompositionTable {
 9      /**
10       * Base relations in RCC8, i.e. the row/column header of the weak
11       * composition table
12       */
13      final String[] baseRCC = { "DC", "EC", "PO", "TPP", "NTPP", "TPPI", "NTPPI"
14      final String[] baseRccInverses = { "DC", "EC", "PO", "TPPI", "NTPPI", "TPP"
15      // final String[] rowColumnHeaderWeakCompTable = { "DC","EC","PO"};
16
17      /**
18       * Save the composition table of weak compositions for all base realations
19       * in RCC8
20       */
21      final String[][] weakCompositionTable = {
22              { "(DC),(EC),(PO),(TPP),(NTPP),(TPPI),(NTPPI),(EQ)", "(DC),(EC),(PO
23                      "(DC),(EC),(PO),(TPP),(NTPP)", "(DC)", "(DC)", "(DC)" },
24              { "(DC),(EC),(PO),(TPPI),(NTPPI)", "(DC),(EC),(PO),(TPP),(TPPI),(EQ
```

Figure 5.12. Interface *CompositionTable*

5.3.4 Results and Output of Query Reformulation

The output of the Adapted Perfect Rewriting algorithm with the proposed optimization is illustrated in Figure 5.13. The inputs of the Query Reformulation process are the TBox from Figure 5.3 and the project conjuctive query from Figure 5.6. The result is a union of conjunctive queries, where every CQ is constructed by two query atoms. The relevant information for the input TBox and input query of the reformulation process are marked respectively within a blue and green rectangular, while the output information (i.e. the output UCQ) is marked within a red rectangular. Important test parameters are printed to the screen in order to be able to compare the computational effectiveness of the Adapted Perfect Rewriting algorithm, evaluated under various scenarios. In addition, it is analyzed how the performance of the algorithm changes by measuring important parameters, such as number of concepts, roles, axioms, PIs and FeaturePaths in the TBox. Furthermore, parameters regarding the size and the nature of the initial search query are also listed and finally the resulting UCQ is displayed, by analyzing its size and origin of the added CQs.

5. Realization

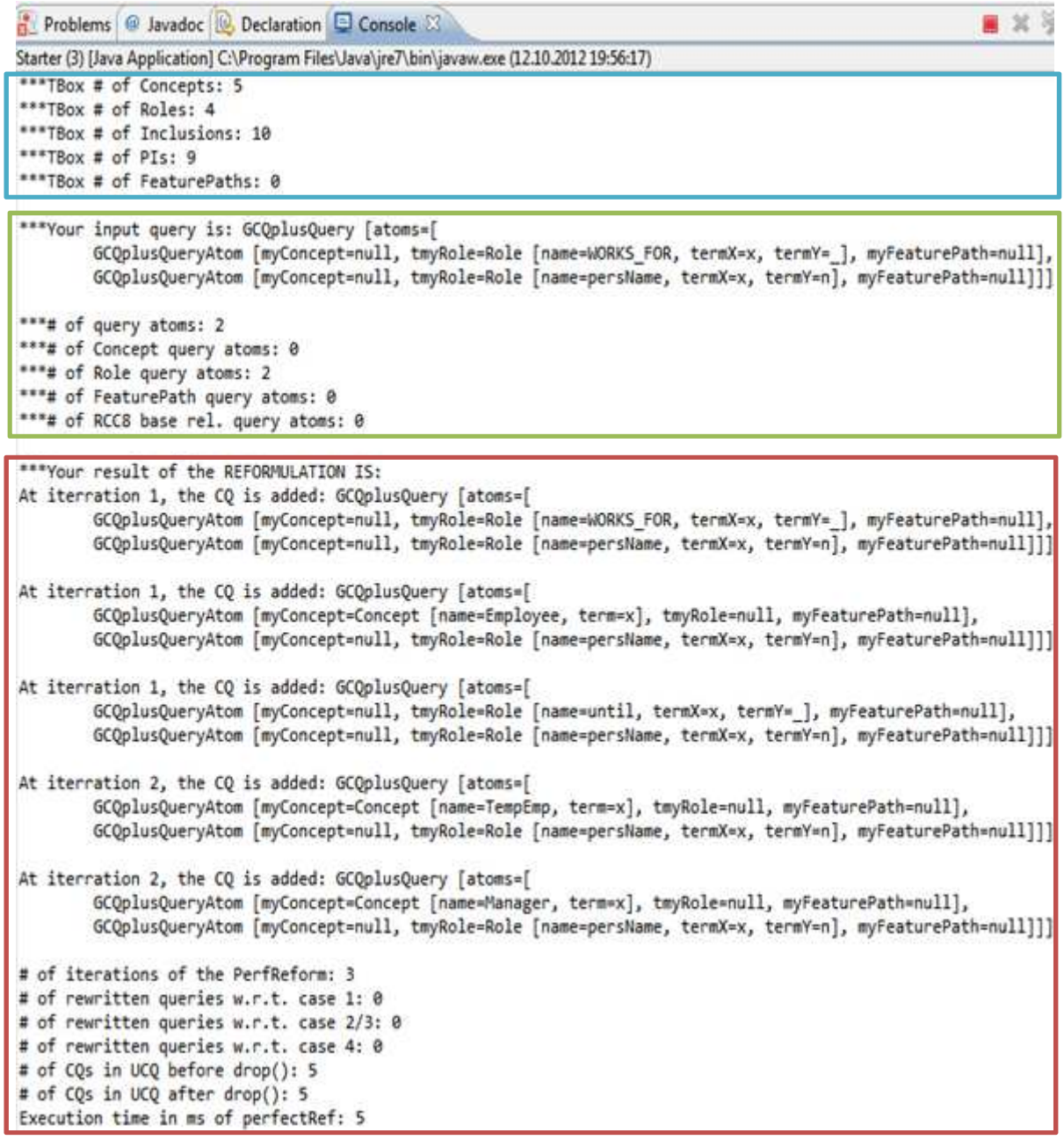

Figure 5.13. Output of The Adapted Perfect Rewriting Algorithm Over a pure DL-Lite Ontology

By comparing the TBox in Figure 5.3 and the TBox test results in Figure 5.13 (blue rectangle), it can be concluded that they match. The same conclusion can be drawn for the query test parameters. The input query *q(x,n) <- WORKS_FOR(x,_) & persName(x,n)* is initially added to the UCQ output result during the first iteration of the algorithm. During the same iteration by applying to the atom *WORKS_FOR(x,_)* the PI *Employee implies (some WORKS_FOR)* and *until implies (some WORKS_FOR)*, the new queries *q(x,n) <- Employee (x) & persName(x,n)* and *q(x,n) <- until(x,_) & persName(x,n)* are respectively added. At the second execution of the while-loop of the algorithm, the query *q(x,n) <- TempEmp (x) &*

persName(x,n) is added according to the application of the PI *TempEmp implies until* to the atom *until(x,_)*. Finally, the new query *q(x,n) <- Manager (x) & persName(x,n)* is added by applying the PI *Manager implies Employee* to the atom *Employee (x)*, thus producing in total 5 conjunctive queries within 5 ms. The same test, using the same TBox and input query, is repeated for the Perfect Rewriting Algorithm, namely the version without considering the spatial modifications, and the measured execution time is 4 ms, which is a negligible difference. However this performance difference is reasonable and expected, since the Adapted Perfect Rewriting algorithm performs an additional check (cf. Figure 5.10, line 6) and further execution of the *drop()* function, which are missing in the original version of the algorithm. This practical test proves that both implementations of the original and adapted versions of the Perfect Rewriting Algorithm have nearly the same performance, when answering queries over pure DL-Lite ontologies.

Another interesting test scenario for the Adapted Perfect Rewriting algorithm is the case when a DL-Lite(RCC8) ontology is considered. The park TBox in Figure 5.4 and the park query in Figure 5.6 are used as inputs for the query reformulation process. The screen output, displaying the output results of the algorithm is presented in Figure 5.14 and Figure 5.15.

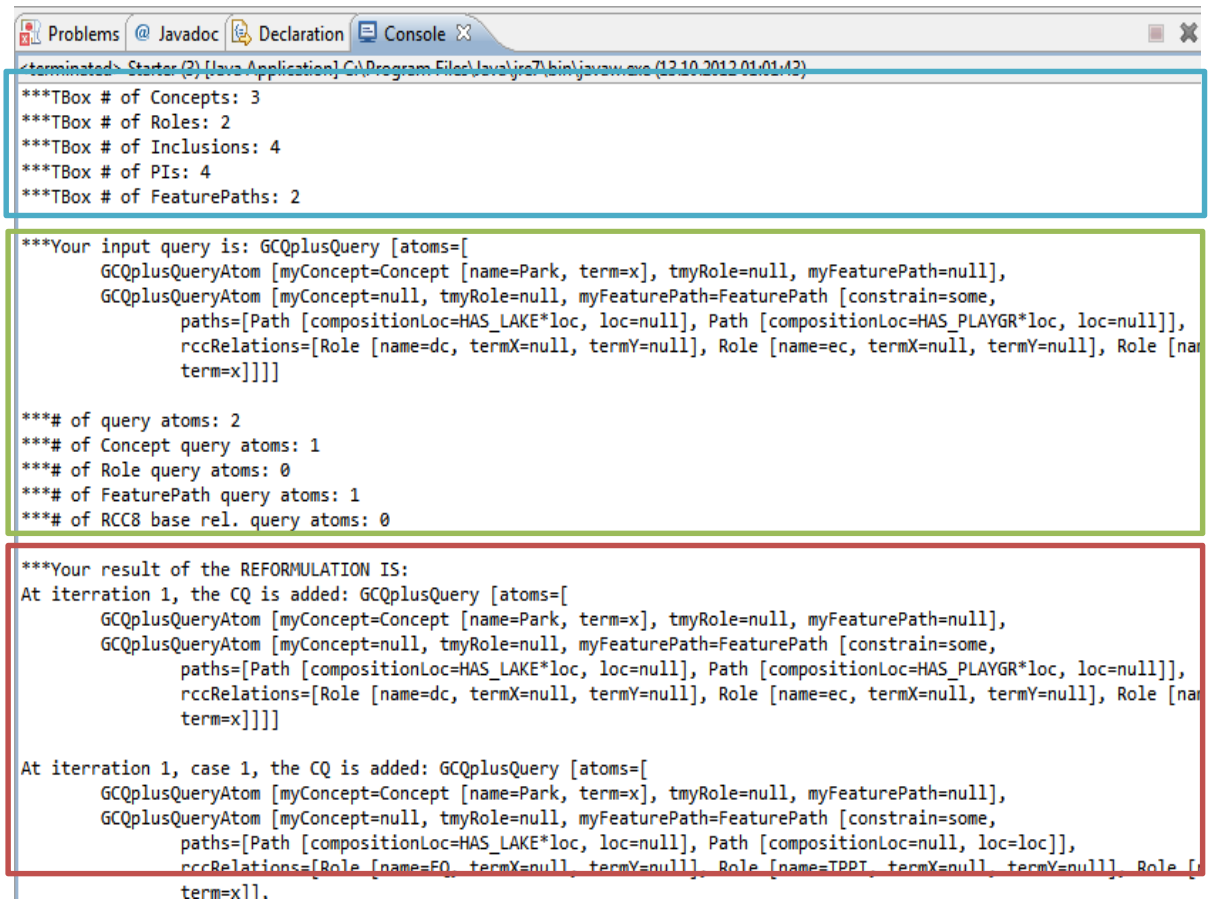

Figure 5.14. Output of The Adapted Perfect Rewriting Algorithm Over a DL-Lite(RCC8) Ontology Part 1

The test parameters of the TBox, input query and queries added during the reformulation process are marked respectively in blue, green and red. During the execution of the Adapted Perfect Rewriting Algorithm 21 CQs are consequently added. Not all of these queries are shown in Figure 5.14, but they are listed in the Appendix C (cf. file Experiment_3_Results.txt). Analyzing Figure 5.15, it can be seen that the The Adapted Perfect Rewriting Algorithm is iterated 4 times in this scenario, by adding 9 new queries (in fact, 18 new query atoms) according to the first rewriting rule in the extended reformulation algorithm, 11 new queries according to the second and third rule and the initial query. After executing the dropping function, only one single query is left, as indicated in Figure 5.15. It is not surprising that the initial input query has two atoms and some of the rewritten queries have three atoms, since the first rewriting rule substitutes one query atom by two new atoms. The entire algorithm for this experiment takes around 48 seconds. The main cause for this execution time is the initial query atom *some HAS_LAKE*loc,HAS_PLAYGR*loc.{dc, ec, po, tpp, tppi, ntppi, eq}(x).* However, 48 seconds is still an acceptable result.

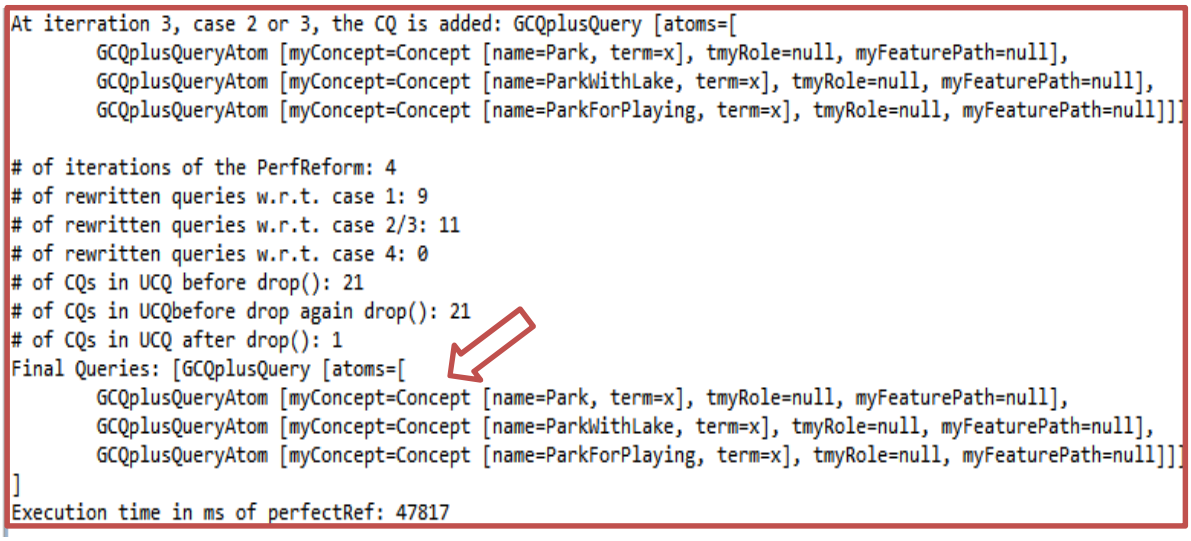

Figure 5.15. Output of The Adapted Perfect Rewriting Algorithm Over a DL-Lite(RCC8) Ontology Part 2

5.4 Unfolding

The Query Unfolding process is part of the QA Reasoner, namely the orange block in Figure 4.1. Before starting the actual unfolding process, the corresponding object-to-data mappings are retrieved from a mapping text file by a parsing algorithm that generates the split versions of the mappings (cf. Chapter 3.2, Figures 3.4 and 3.6) and translates them into Java objects. The classes, representing the mapping assertions are stored in the package *mappings* and they are illustrated in Figure 5.16. The output of the parsing procedure is an object of type *MappingBox*, consisting of a list of objects of type *ObjectToDataMapping*.

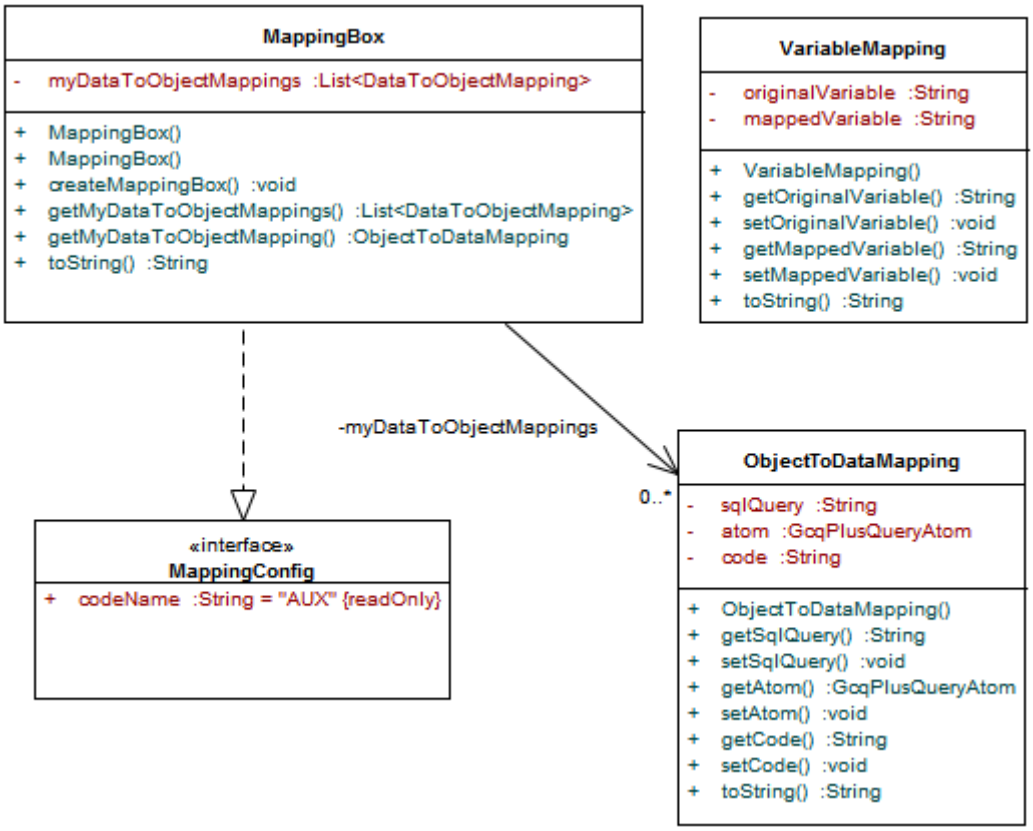

Figure 5.16. UML Class Diagrams of Package *mappings*

The class *ObjectToDataMapping* encapsulates a query *atom* of type *GcqPlusQueryAtom*, a *sqlQuery* string and a logic programming clause *code*. Theoretically, as described in [7, p. 339], this clause denotes the result of the evaluation over the database of the SQL query, that is encapsulated in the left hand-side of the mapping (i.e. the attribute *sqlQuery*). The *code* attribute contains a String that has the form AUX_{A_B}, where the indexes A and B are numbers. The index A is used in order to identify that two split mappings of type *ObjectToDataMapping* stem from the same root mapping assertion, i.e. they have identical SQL string and *GcqPlusQueryAtom* objects. The index B is an increment counter, showing the position of the corresponding *ObjectToDataMapping* element in the attribute list *myDataToObjectMappings* from the class *MappingBox*. For instance, if an object X and object Y both of type *ObjectToDataMapping* have respectively code AUX_{1_2} and AUX_{1_5} , this fact infers that X and Y refer to the same mapping assertion and X is the 2^{nd} and Y is the 5^{th} element from the list *myDataToObjectMappings*. The actual unfolding of the UCQ, being the result of the Query Reformulation step, is done in the method *unfold( )*, where by executing the operations *getSubstitutes( )* and *findAllCombinations( )*, each of the atoms of the UCQ are unified in all possible ways with the corresponding *code* attributes from the list *myDataToObjectMappings*, by producing a new list of *UnfoldedQuery* objects, returned by the method unfold().

The variable terms of the atoms from the right hand-side of the mapping assertions and the variables from the input UCQ, resulting from the Query Reformulation process are not

syntactically identical, even though that they have the same meaning. In addition, the former are not only variables, but they can be also variable terms, containing function symbols. That is the reason why an automatic variable matching algorithm is also implemented, since it can happen that during the unification step some of the code combinations result in queries, which are not any more dependent on the initial head variables of the corresponding query or contain only unbound variables. These malformed queries should be filtered. The variable matching is realized with the help of the classes *VariableMapping* (cf. Figure 5.16).

The final step of the Unfolding process is realized by the method *createSQLquery(...)* from the abstract class *SqlQueryGeneration* by substituting each AUX_{A_B} predicate from the *UnfoldedQuery* list with its SQL equivalent, thus creation a new final SQL query that is a union of select-project-join queries.

In order to illustrate and test the implementation of the unfolding, the resulting UCQ from the query reformulation in Figure 5.13 is used. A sample file of mapping assertions [7, p. 334], presented in Figure 5.17, maps the objects from the TBox in Figure 5.3 to the data in the database in Figure 5.18.

```
SELECT "D1"."SSN", "D1"."PROJ", "D1"."D" FROM "D1" ->
TempEmp(pers["D1"."SSN"]) &
WORKS_FOR(pers["D1"."SSN"],proj["D1"."PROJ"]) &
projName(proj["D1"."PROJ"],"D1"."PROJ") &
until(pers["D1"."SSN"],"D1"."D")
;;;
SELECT "D2"."SSN", "D2"."NAME" FROM "D2" ->
Employee(pers["D2"."SSN"]) & persName(pers["D2"."SSN"],"D2"."NAME")
;;;
SELECT "D4"."SSN", "D3"."NAME" FROM "D3","D4" WHERE
"D3"."CODE"="D4"."CODE" -> Manager(pers["D4"."SSN"]) &
persName(pers["D4"."SSN"],"D3"."NAME")
;;;
SELECT "D3"."CODE", "D3"."NAME" FROM "D3" WHERE "D3"."CODE"
NOT IN (SELECT "D4"."CODE" FROM "D4") ->
Manager(mgr["D3"."CODE"]) &
persName(mgr["D3"."CODE"],"D3"."NAME")
```

Figure 5.17. Sample Mapping Assertions File *mappingsProject.txt*

In order to realize a correct parsing, then the object-to-data mappings should be specified accordingly, by taking into account that the SQL SELECT key-word should be always capitalized and every assertion should be written on a single line, since the Parser reads the text file line by line and any additional white-space characters can result in unsuccessful parsing.

D1 *[ID: bigint; SSN: character varying(10); PROJ: character varying(50); D:date]*

D2 *[ID: bigint; SSN: character varying(10); NAME: character varying(50)]*

D3 *[ID: bigint; CODE: character varying(10); NAME: character varying(50)]*

D4 *[ID: bigint; CODE: character varying(10); SSN: character varying(10)]*

Figure 5.18 Table Signatures of a Sample Project Database

The tables in Figure 3.3 store information about projects and employees, where ID is always the primary key for the corresponding table. Table D1 stores temporary employees, their project names and end dates, while table D2 matches a social security number of an employee to his name. The table D3 store managers, while D4 relates managers' codes with their SSNs.

The output of the Unfolding step is illustrated is Figure 5.19, where it can be seen that the UCQ from Figure 5.13 and the input search query *q(x,n)<-WORKS_FOR(x,_) & persName(x,n)* have produced 6 distinct *UnfoldedQuery* objects, revealed within a blue rectangle and the unfolding procedure took 35 ns. As a result the green rectangular block shows that the *UnfoldedQuery* objects are translated to a final SQL query, consisting of 6 unions of select-project-join queries. Finally, this SQL statement is directly issued over the database and its results set is the result of the Unfolding step and thus also the outcome of the entire Ontology Based Query Answering Process is displayed within the red rectangle. The tuples *(mgr[code_2], "Moeller"), (pers[55555], "OEZCEP")* and *(pers[12345], "GUDOV")* are the requested certain answers.

5. Realization

```
------------------------------Start of Unfolding--------------------------------------
***# of split mappings: 10
Unfolded Queries:
[Query [head=[pers["D1"."SSN"], "D2"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX1_1, AUX2_5], sqlJoins="D1"."SSN"="D2"."SSN" ]
, Query [head=[pers["D1"."SSN"], "D3"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX1_1, AUX3_7], sqlJoins="D1"."SSN"="D4"."SSN" ]
, Query [head=[pers["D2"."SSN"], "D2"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX2_4, AUX2_5], sqlJoins=]
, Query [head=[pers["D2"."SSN"], "D3"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX2_4, AUX3_7], sqlJoins="D2"."SSN"="D4"."SSN" ]
, Query [head=[pers["D4"."SSN"], "D3"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX3_6, AUX3_7], sqlJoins=]
, Query [head=[mgr["D3"."CODE"], "D3"."NAME"]]
        UnfoldedQuery [codeCombinations=[AUX4_8, AUX4_9], sqlJoins=]
]
***# of SQL union queries: 6
Execution time in ms of the Unfolding: 35
```

```
SQL query:
SELECT  CONCAT ( CONCAT ( 'pers[' ,"D1"."SSN"),']'),"D2"."NAME"
FROM "D1","D2"
WHERE "D1"."SSN"="D2"."SSN"
UNION
SELECT  CONCAT ( CONCAT ( 'pers[' ,"D1"."SSN"),']'),"D3"."NAME"
FROM "D1","D3","D4"
WHERE "D1"."SSN"="D4"."SSN"  AND ("D3"."CODE"="D4"."CODE" )
UNION
SELECT  CONCAT ( CONCAT ( 'pers[' ,"D2"."SSN"),']'),"D2"."NAME"
FROM "D2"
UNION
SELECT  CONCAT ( CONCAT ( 'pers[' ,"D2"."SSN"),']'),"D3"."NAME"
FROM "D2","D3","D4"
WHERE "D2"."SSN"="D4"."SSN"  AND ("D3"."CODE"="D4"."CODE" )
UNION
SELECT  CONCAT ( CONCAT ( 'pers[' ,"D4"."SSN"),']'),"D3"."NAME"
FROM "D3","D4"
WHERE ("D3"."CODE"="D4"."CODE" )
UNION
SELECT  CONCAT ( CONCAT ( 'mgr[' ,"D3"."CODE"),']'),"D3"."NAME"
FROM "D3"
WHERE ("D3"."CODE" NOT IN (SELECT "D4"."CODE" FROM "D4") )
```

```
mgr[code_2]     Moeller
pers[55555]     OEZCEP
pers[12345]     GUDOV
***# of results tuples: 3
Execution time in ms of the SQL answering: 29
```

Figure 5.19. Unfolding Sample Output

6 Evaluation

6.1 Final Results and Experiments Discussion

The results, presented in Chapter 5.3.4, as well as the outcomes of further experiments are listed in Figure 6.1. The input files for the experiments are provided in Appendix C

Test parameter	**Exp. 1/ Result**	**Exp. 2/ Result**	**Exp. 3/ Result**	**Exp. 4/ Result**	**Exp. 5/ Result**
Tbox					
concepts	5	5	3	5	5
roles	4	4	2	4	4
axioms	10	10	4	8	8
PIs	9	9	4	8	8
FeaturePaths	0	0	2	4	4
Input Query					
atoms	2	2	2	2	3
concept atoms	0	0	1	1	2
role atoms	2	2	0	0	0
FeaturePath atoms	0	0	1	1	1
Perfect Rewr. Algorithm					
version	adapted	original	adapted	adapted	adapted
iterations	3	3	4	4	4
added queries case 1	0	x	9	9	9
added queries case 2/3	0	x	11	11	11
added queries case 4	0	x	0	0	0
queries before drop( )	5	x	21	21	21
queries after drop( )	5	x	1	1	1
execution time in ms	5	4	47817	47985	48711

Figure 6.1. Consolidated Experiment Results for Perfect Query Rewriting over Ontologies

Analyzing the experiments in Figure 6.1, it is obvious that the original Perfect Rewriting algorithm performs better in pure DL-Lite ontologies, comparing Experiment 1 with the last three experiments. However, this is not surprising, since when pure DL-Lite ontologies are considered, the Adapted Perfect Rewriting Algorithm should behave as the original one. An interesting outcome is the execution time of the adapted algorithm in experiment 3 and 4, where the size of the TBox is increased and the $time_{exp4} > time_{exp3}$. This increase in time is negligible, but expected, since by adding new concept, roles, etc. then the algorithm executes more comparisons operations in the search methods. The increase of the atoms in the input query of experiment 5 has also a negative effect on the execution time. However, it should be clear that experiments 3, 4 and 5 only test the dependency of increasing the number of

concepts and roles for the TBox and the query. That is the reason it is recommended that the spatial part of the algorithm is tested in further experiments, by increasing the number of FeatureParth atoms both in the Tbox and the query, i.e. adding atoms of the form $\exists U_1,U_2.r$. Further recommended experiments are to evaluate the entire query answering process when the number of mappings is increased. Finally, database experiments can be also performed by increasing the length of the output select-project-join SQL query and the size of the database.

6.2 Future Research and Recommendations

The system *Parser* from Figure 4.1 can be improved by making use of XML files instead of text files for the TBoxes, input queries and object-to-data mappings, since an XML file can be automatically validated against a predefined XML schema and translated to Java objects with the help of JAXB bindings. This modification will both reduce the error rate of wrong user inputs and the complexity of the *Parser*. Furthermore, it is better to realize the query Reformulation and Unfolding processes as threads, i.e. classes *QueryReformulation* and *QueryUnfolding* (cf. Figure 4.2) in order to provide parallel computing possibilities. In the developed prototype of the system, it can happen that the query answering process takes a longer time and the GUI freezes for this period of time. This is not a desirable feature and it can this issue can be eliminated by putting the main computation process into a thread. From a usability point of view, it is also recommended to enrich the Graphical User Interface, so the user can have more possibilities to interact with the system without directly modifying the source code. A user friendly program should also encapsulate *JProgressBar* elements (i.e. progress bars or download boxes) in order to illustrate how many percent of the query answering process have been accomplished.

The developed system prototype does not understands GCQ^+ query atoms of the form $r(x^*, y^*)$, $r^* \in Rel_{RCC8}$ *and* $x^*,y^* \in Ax_{RCC8}$. Currently if these atoms are used then the program identifies them as standard role without taking into account their spatial properties. However, a slight modification should be done and respectively tested. The first proposal is not to change the Rewriting and Unfolding parts of the *Reasoner* but try to use SQL spatial correspondence functions, which can handle relations of the type DC(x*,y*), TPP(x*,y*), etc. and embed these mappings in the mapping text files. The OpenGIS functions for geometry relationship [38] are possible alternatives, e.g. ST_Equals(geometry, geometry), ST_Touches(geometry, geometry), etc. However, a more reasonable approach is to include the latter mappings, e.g. in a Java interface, since these mappings are universal and the user does not need to specify them explicitly. Thus the Undolding process should be modified in order to correctly insert openGIS functions into the SQL query.

Another reasonable improvement is to migrate the developed system to an online platform that can be accessed by many users without needing to distribute the source code and thus also benefiting from the advantages of a web application versus desktop standalone program, e.g. less operational cost for software installation and maintenance, better system accessibility

and availability, immediate effect of software patches, etc. In addition, it will be beneficial if additional features and reasoning tasks are added to the developed system, e.g. consistency checks and satisfiability tests.

7 Conclusion

The core theoretical conclusions of this Master Thesis is that the query language GCQ^+, capable of querying spatial ontologies with respect to a DL-Lite(RCC8) TBox, can be used to answer complex queries in an effective way over geographical databases. This is realized by the integration of a realistic top-down approach that makes use of object-to-data mappings and an optimized adapted version of a Perfect Rewriting Algorithm, considering geographic application domains.

In a nutshell, at the first step the input query, containing GCQ^+ atoms, is reformulated with the help of the Positive Inclusions (PIs) from the DL-Lite(RCC8) TBox. After the Perfect Rewriting processing, the second step performs an Unfolding algorithm that avoids the explicit materialization of the virtual ABox and consequently generates an SQL query that can be issued directly over the spatial database. This is achieved by using the object-to-data mapping specifications. Thus, although the described reasoning algorithms are dependent on the size of the TBox and the queries, this is not a crucial disadvantage, since normally w.r.t. geographical scenarios, the size of the TBox and queries is relatively small in comparison to the size of the persistent geo-data, stored in a database. The main asset of this approach is that the result set of the output SQL query of the Unfolding step coincides with the results of the initial query over the ontology. Thus the data complexity of the entire algorithm is in AC^0, making use of the query optimization techniques provided by current database management systems.

Last but not least, a software application, called OnQuAnSpatial has been developed that realizes the latter mentioned theoretical conclusion and provides the basic features of ontology based query answering over spatial databases. Although, the developed system is not a fully functional and bug-free commercial product, it usage is highly recommend. The OnQuAnSpatial prototype can be successfully applied for educational purposes, e.g. in workshops and practical exercises for students, attending lectures in Artificial Intelligence or Computer Logics. Moreover, this application can be integrated into research projects, dealing with ontology based query answering over spatial or traditional databases. In addition, the OnQuAnSpatial system can be used in order to practically test the effectiveness and computational complexity of the Adapted Perfect Rewriting Algorithm over spatial ontologies, by defining worst-case scenarios and changing various test parameter dependences such as size and type of the TBox, size of the input query and size of the database.

Appendix

A. References

[1] Nardi, D; Brachman, R. J., "An Introduction to Description Logics," in *THE DESCRIPTION LOGIC HANDBOOK*, F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, Eds., Cambridge, Cambridge University Press, 2003.

[2] F. Wolter, "Ontology Languages," 19 Sep 2011. [Online]. Available: http://www.csc.liv.ac.uk/~frank/teaching/comp08/lecture1.pdf. [Accessed 1 Sep 2012].

[3] Baader, F.; Nutt, W., "Basic Description Logics," in *THE DESCRIPTION LOGIC HANDBOOK*, F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, Eds., Cambridge, Cambridge University Press, 2003, p. 43.

[4] Brachman, R. J.; Levesque, H. J., "The Tractability of Subsumption in Frame-Based Description Languages," AAAI-84 Proceedings. Copyright ©1984, AAAI (www.aaai.org), California, 1984.

[5] Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G; Lenzerini, M; Rosati, R, "Linking Data to Ontologies," Rome, 2008.

[6] D. Calvanese, "Knowledge Representation and Ontologies, Part 1: Modeling Information through Ontologies," Free University of Bolzano/Bozen, 09 July 2012 / kro_1. [Online]. Available: http://www.inf.unibz.it/~calvanese/teaching/11-12-kro/lecture-notes/KRO-1-ontologies-uml.pdf. [Accessed 25 08 2012].

[7] Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rodriguez-Muro, M.; Rosati, R., "Ontologies and Databases:The DL-Lite Approach," Springer, 2009.

[8] The PostgreSQL Global Development Group , "PostgreSQL," The PostgreSQL Global Development Group, 2012. [Online]. Available: http://www.postgresql.org/. [Accessed 6 10 2012].

[9] Oracle, "Oracle® Spatial Developer's Guide 11g Release 1 (11.1)," Oracle, 2009. [Online]. Available: http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/toc.htm. [Accessed 7 10 2012].

[10] Microsoft, "Working with Spatial Data (Database Engine) SQL Server 2008 R2," Microsoft, 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb933876.aspx. [Accessed 8 10 2012].

[11] IBM, "DB2 Spatial Extender and Geodetic Data Management Feature User's Guide and Reference," 2009. [Online]. Available: ftp://public.dhe.ibm.com/ps/products/db2/info/vr9/pdf/letter/en_US/db2sbe90.pdf. [Accessed

8 10 2012].

[12] United States Census Bureau, "TIGER/Line® Shapefiles and TIGER/Line® Files," 16 Aug 2012. [Online]. Available: http://www.census.gov/geo/www/tiger/shp.html. [Accessed 8 Oct 2012].

[13] I. Horrocks, "The FaCT System," , 2003. [Online]. Available: http://www.cs.man.ac.uk/~horrocks/FaCT/. [Accessed 8 Oct 2012].

[14] P. F. Patel-Schneider, "The DLP Experimental Description Logic System and Propositional Modal Logic Satisfiability Checker," June 2010. [Online]. Available: http://ect.bell-labs.com/who/pfps/dlp/. [Accessed 8 Oct 2012].

[15] Moeller,R.; Haarslev,V., "Racer - Renamed Abox and Concept Expression Reasoner," 2012. [Online]. Available: http://www.sts.tu-harburg.de/~r.f.moeller/racer/. [Accessed 06 Oct 2012].

[16] QuOnto, "QuOnto QUerying ONTOlogies," SAPIENZA Università di Roma --- Dipartimento di Informatica e Sistemistica Antonio Ruberti, 2012. [Online]. Available: http://www.dis.uniroma1.it/quonto/?q=node/25. [Accessed 8 Oct 2012].

[17] Smid, M.; Kouba, Z., "OnGIS: Ontology Driven Geospatial Search and Integration," 18 Sep 2012. [Online]. Available: http://ceur-ws.org/Vol-901/paper3.pdf. [Accessed 8 Oct 2012].

[18] M. Vardi, "The Complexity of Relational Query Languages," 1982.

[19] J. Savage, "Models of Computation, Exploring the Power of Computing," 2008.

[20] S. Abitebou, R. Hull and V. Vianu, "Chapter 17. First Order, Fixpoint, and While," in *Foundations of Databases*, Addison-Wesley, 1995, pp. 430-433.

[21] Eiter,t.; Gottlob, G.; Ortiz, M.;Simkus, M., "Query Answering in the Description Logic Horn-SHIQ," 2008.

[22] Calvanese, D.;Zakharyaschev, M., "22nd European Summer School in Logic Language and Information," Aug 2010. [Online]. [Accessed 16 Sep 2012].

[23] T. Pankowski, "XML Schema Mappings Using Schema Constraints and Skolem Functions," in *Knowledge-Driven Computing. Knowledge Engineering and Intelligent Computations*, Springer-Verlag Berlin Heidelberg, 2008, pp. 199-216.

[24] Barrasa, J.; Corcho, O.; Gomez-Perez, A., "R2O, an extensible and semantically based database-to-ontology mapping language," 2004. [Online]. Available: http://www.cs.man.ac.uk/~ocorcho/documents/SWDB2004_BarrasaEtAl.pdf. [Accessed 17 Sep 2012].

[25] Goasdoue, F.; Lattes, V.; Rousset, M.-C., "The use of CARIN language and algorithms for information integration: The Picsel system," 2000. [Online]. Available: http://www.lri.fr/~goasdoue/bib/GoasdoueLattesRousset-IJCIS-00.pdf. [Accessed 17 Sep 2012].

Appendix

[26] Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Rosati, R., "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family," 2007. [Online]. Available: http://www.dis.uniroma1.it/~degiacom/papers/2007/calv-etal-JAR-2007.pdf. [Accessed 18 Sep 2012].

[27] Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Rosati, R., "Path-based Identification Constraints in Description Logics," 2008. [Online]. Available: http://www.dis.uniroma1.it/~degiacom/papers/2008/KR08ids.pdf. [Accessed 18 Sep 2012].

[28] Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Rosati, R., "Data Complexity of Query Answering in Description Logics," 2006. [Online]. Available: http://www.dis.uniroma1.it/~degiacom/papers/2006/KR06Calvanese.pdf. [Accessed 18 Sep 2012].

[29] Oezcep, O.L.; Moeller, R., "Computationally Feasible Query Answering over Spatio-thematic Ontologies," 2012. [Online]. Available: http://www.sts.tu-harburg.de/~r.f.moeller/papers/2012/OeMo12.pdf. [Accessed 19 Sep 2012].

[30] Oezcep, O.L.; Moeller, R., "Combining DL-Lite with Spatial Calculi for Feasible Geo-thematic Query Answering," 2012.

[31] Randell, D. A. ; Cui,Z.; Cohn, A. G;, "A Spatial Logic based on Regions and Connection," PROCEEDINGS 3RD INTERNATIONAL CONFERENCE ON KNOWLEDGE REPRESENTATION AND REASONING, 1992.

[32] J. Renz, Qualitative Spatial Reasoning with Topological Information, vol. 2293, Wien: Springer, 2002.

[33] Oezcep, O.L; Moeller, R., "Combining Lightweight Description Logics with the Region Connection Calculus," 2011.

[34] Google Maps, "Google Maps Europe," Google, 2012. [Online]. Available: http://maps.google.com/. [Accessed 2012 Sep 21].

[35] Oezcep, O.L.; Moeller, R., "Scalable geo-thematic query answering," Institute for Softwaresystems (STS), Hamburg University of Technology, Hamburg, 2012.

[36] D. Calvanese, "Knowledge Representation and Ontologies Part 3: Query Answering in Databases and Ontologies," 2012.

[37] D. Calvanese, "Knowledge Representation and Ontologies Part 5: Reasoning in the DL-Lite Family," 2012.

[38] PostGIS, "Chapter 6. PostGIS Reference," 2012. [Online]. Available: http://postgis.refractions.net/documentation/manual-1.3/ch06.html#id2574404. [Accessed 1 Oct 2012].

Appendix

[39] Brachman, R. J.; Schmolze, J. G., "An Overview of the KL-One Knowledge Representation System," 1985. [Online]. Available: http://eolo.cps.unizar.es/docencia/MasterUPV/Articulos/An%20Overview%20of%20the%20KL-ONE%20Knowledge%20Representation%20System-Brachman1985.PDF. [Accessed 2 Sep 2012].

[40] M. Schmidt-Schauß, "Subsumption in KL-ONE is Undecidable," 1989. [Online]. Available: http://www.ki.informatik.uni-frankfurt.de/papers/schauss/KLONE-UNDEC2004.pdf. [Accessed 2 Sep 2012].

[41] Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Rosati, R., "DL-Lite: Tractable description logics for ontologies," 2005.

[42] PostGIS, "Chapter 6. PostGIS Reference," 2012. [Online]. Available: http://postgis.refractions.net/documentation/manual-1.3/ch06.html#id2574404. [Accessed 1 10 2012].

Appendix

B. Abbreviations

A
AWT - Abstract Window Toolkit

B

C

D
DBMS – Database Management System
DL – Description Logics

E

F
FOL – First Order Logic

G
GCIs - General Inclusion Axioms
GPA - Grade Point Average

H

I

J
JAXB - Java Architecture for XML Binding

K
KB- Knowledge Base

L

M
MAF - Master Address File

N

O
OBDA – Ontology Based Data Access
OnQuAnSpatial - Ontology Based Query Answering over Spatial Databases
OWL - Web Ontology Language

P

Q

R
R_2O - Relational to Ontology language
RCC - Region Connection Calculus

Appendix

R
RDMS - Relational Database Management System

S
SQL - Structured Query Language

T
TIGER - Topologically Integrated Geographic Encoding and Referencing

U
UCQ – Union of Conjunctive Queries
UML- Unified Modeling Language
UNA – Unique Name Assumption

V
v/r – value restriction

W
W3C - World Wide Web Consortium

X
XML - Extensible Markup Language

Y

Z

Appendix

C. CD Content

The CD attached to this Master Thesis contains:

1. A PDF document of this Master Thesis
2. The complete java source code of the developed OnQuAnSpatial system
3. Required Java libraries to run the application
4. Experiments and test results