CHOROS: A Reasoning and Query Engine for Qualitative Spatial Information

Georgios Christodoulou

Department of Electronic and Computer Engineering
Technical University of Crete

2011

Acknowledgments

This thesis would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Associate Professor, Euripides G.M Petrakis who was abundantly helpful and offered invaluable assistance, support and guidance.

Also, I would like to thank Professor Minos Garofalakis and Assistant Professor Michail G. Lagoudakis who agreed to evaluate my diploma thesis.

I would like to thank the authority of Technical University of Crete(TUC) for providing us with a good environment and facilities to complete this project.

Special thanks should be given to my laboratory colleagues who helped me in many ways. Finally, words alone cannot express the thanks I owe to my family and friends, for their understanding and encouragement.

Abstract

In this thesis, we present CHOROS, a qualitative spatial reasoning engine implemented in Java. CHOROS provides consistency checking and query answering over spatial data represented with the Region Connection Calculus (RCC) and the Cone-Shaped directional logic formalism (CSD). It supports all RCC-8 and CSD-9 relations as well as standard RDF/OWL semantic relations, both represented in RDF/OWL. As such, it can answer mixed SPARQL queries over spatial and non-spatial relation types. CHOROS extends PelletSpatial's [4] hybrid architecture, which is based on a composition table that implements a path-consistency algorithm. We also introduce a multithreading technique that enables to execute CSD and RCC consistency checking concurrently. As a case study and to objectively assess the performance of CHOROS we developed the "TUC spatial ontology" providing a qualitative spatial description of the regions, forming the campus of Technical University of Crete (TUC). Finally, we discuss and evaluate possible optimizations of CHOROS and compare its performance with that of a spatial reasoner implemented in SWRL and runs under Protégé [2].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Qualitative reasoning is an approach for dealing with commonsense knowledge without using numerical computation. Instead, one tries to represent knowledge using a limited vocabulary such as qualitative relationships between entities or qualitative categories of numerical values. An important motivation for using a qualitative approach is that it is considered to be closer to how humans represent and reason about commonsense knowledge. Another motivation is that it is possible to deal with incomplete knowledge.

A very important concept of commonsense knowledge is space. Because of the richness of space and its multiple aspects, however, most work in qualitative spatial reasoning has focused on single aspects of space. Two of the most important aspects of space are topology and orientation.

1.1 Problem Definition

It is a common practice to use Web Ontology Language (OWL) ontologies to describe spatial regions and relations between these regions, such as relative directional position or spatial containment and overlap. However, it is not possible to directly encode the semantics of these relations using the expressivity of OWL and the Description Logics (DL) that OWL is based on. As a consequence, there might be inconsistencies in spatial relations that will not be detected by an OWL reasoner or an OWL reasoner might not return all the answers to spatial queries since it cannot compute all spatial inferences.

1.2 Background

The most popular reasoning methods used in qualitative spatial reasoning are constraint based techniques adopted from previous work on temporal reasoning [12, 13]. Reasoning applies on sets of qualitative spatial relations which are jointly exhaustive and pairwise disjoint, i.e., between any two spatial entities exactly one of the basic relations holds. The set of all possible relations is then the set of all possible unions of the basic relations. Reasoning is realized by exploiting composition of relations. For instance, if the binary relation R1 holds between entities A and B and the binary relation R2 holds between B and C, then the composition of R1 and R2 restricts the possible relationship between

A and C. Compositions of relations are usually pre-computed and stored in a composition table [14].

The SOWL [7] spatial representation implements reasoning rules for RCC-8 relations and cone-shaped direction relations using SWRL and OWL 2.0 property axioms. Path consistency is implemented by introducing rules defining compositions and intersections of supported relations until a fixed point is reached or until an inconsistency is detected. Reasoners that support DL-safe rules such as Pellet [5, 6] can be used for inference and consistency checking over spatio-temporal relations.

PelletSpatial [4] extends Pellet OWL reasoner with qualitative spatial reasoning capabilities. It supports checking the consistency of spatial relations expressed using RCC-8 and computes new spatial inferences from asserted relations. The spatial relations are expressed in RDF/OWL and and can be applied on arbitrary domain ontologies. PelletSpatial implements two RCC reasoner components: (a) A reasoner implementing the translation of RCC relations to OWL-DL class axioms while preserving their semantics and (b) a reasoner operating on the RCC composition table implementing a path-consistency algorithm [9].

1.3 Proposed Work

In this thesis, we present CHOROS which supports consistency checking and query answering over spatial data represented with the Region Connection Calculus (RCC) and the Cone-Shaped directional logic formalism (CSD). In that respect, it extends PelletSpatial to support CSD-9 relations in addition to RCC-8 relations as well as standard RDF/OWL semantic relations, both represented in RDF/OWL. As such, it can answer mixed SPARQL queries over spatial and non-spatial relation types.

In our work we implement path-consistency algorithm 3.4, in Java, based on the the composition tables of 3.1 for RCC-8 and 3.2 for CSD-9 which are also implemented in SOWL. As Java supports multithreading natively, we introduce a multithreading technique that enables "parallel" execution of CSD and RCC consistency checking.

1.4 Thesis Outline

Background knowledge and related research are discussed in Chapter 2. A description of ontologies and of qualitative spatial calculi is presented. Basics of reasoning are introduced as well. In Chapter 3 we discuss issues on the representation of spatial information in RDF/OWL. CHOROS architecture and reasoning are presented in the first two sections of Chapter 3 along with possible optimizations in Section 3.3. Chapter 4 presents the results from an empirical investigation of the practical efficiency of CHOROS. In particular, in Section 4.2 we introduce the "TUC spatial ontology" which we use as a case study for assessing the performance of CHOROS. Finally, conclusions and issues for further research are discussed in Chapter 5.

Chapter 2

Related Work

2.1 Ontologies-OWL

Ontologies are used to capture knowledge on a domain of interest. An ontology describes the concepts of the domain and also the relationships that hold between those concepts. By defining shared and common domain theories, ontologies help both people and machines to communicate concisely, supporting the exchange of semantics and not only syntax. In recent years, ontologies been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. In nowadays, they are used in applications such as scientific knowledge portals, information management and integration systems, electronic commerce, and semantic web services.

In Semantic Web the information contained in documents is given an explicit meaning, making it easier to be processed by applications. OWL [1] can be used to represent the meaning of terms and the relationships between those terms. It is more expressive than XML, RDF and RDF-S, making it easier to represent machine interpretable content on the Web. OWL ontologies may be categorized into three species or sublanguages: OWL-Lite, OWL-DL and OWL-Full. OWL 2 [15] is an extension and revision of the OWL, developed by the W3C Web Ontology Working Group and published in 2004. OWL 2 adds new functionality with respect to OWL. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressivity [18].

Languages of the OWL family are capable of creating classes, properties, defining instances and its operations. An instance is an object. It corresponds to a description logic individual. A class is a collection of objects. It corresponds to a description logic [17] concept. A property is a directed binary relation that specifies class characteristics. It corresponds to a description logic role. Datatype properties are relations between instances of classes and RDF literals or XML schema datatypes. Object properties are relations between instances of two classes. OWL also supports various operations on classes such as union, intersection and complement as well as class enumeration, cardinality, and disjointness.

2.2 Reasoning

A semantic reasoner, reasoning engine, rules engine, or simply a reasoner, is a piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalizes that of an inference engine, by providing a richer set of mechanisms to work with. The inference rules are commonly specified by means of an ontology language, and often a description language.

2.2.1 Pellet

Pellet [5, 6] is a complete OWL reasoner with very good performance and a number of unique features. It is written in Java and is open source under a very liberal license. It has been adopted in projects and application from pure research to industrial settings. Pellet supports reasoning with the full expressivity of OWL-DL (SHOIN(D) in Description Logic jargon) and has been extended to support the more recent OWL 2 specification (SROIQ(D)). It provides all the standard inference services that are traditionally provided by DL reasoners:

- *Consistency checking:* Ensures that an ontology does not contain any contradictory facts. The OWL 2 Direct Semantics provides the formal definition of ontology consistency used by Pellet.
- *Concept satisfiability:* Determines whether its possible for a class to have any instances. If a class is unsatisfiable, then defining an instance of that class makes the entire ontology inconsistent.
- *Classification:* Computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.
- *Realization:* Finds the most specific classes that an individual belongs to; i.e., realization computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with respect to a class hierarchy. Using the classification hierarchy, it is also possible to get all the types for each individual.

Pellet relies on an implementation of a direct tableau algorithm [19, 20] for a DL-safe rules extension to OWL-DL. This implementation allows one to load and reason with DL-safe rules encoded in SWRL [16] and includes support for some SWRL built-ins.

2.2.2 SWRL

The Semantic Web Rule Language (SWRL) [16] is an expressive OWL-based rule language. SWRL allows users to write rules that can be expressed in terms of OWL concepts to provide more powerful deductive reasoning capabilities than OWL alone. Semantically, SWRL is built on the same description logic foundation as OWL does and provides similar strong formal guarantees when performing inference.

2.3 Qualitative Spatial Models

The qualitative approach to spatial as well as to temporal information is popular in Artificial Intelligence and related research fields [21, 22]. This is mainly because precise numerical information is often unavailable or not necessary in many real world applications. Typically, the qualitative approach represents spatial information by introducing a (binary) relation model on the universe of spatial entities, which contains a finite set of binary relations defined on the universe. Finding a proper relation model, or a qualitative calculus, is the key to the success of the qualitative approach to spatial reasoning. In the past twenty years, dozens of spatial relation models have been developed. Since relations in the same model are ideally homogenous, most spatial calculi focus on one single aspect of space, e.g. topology, direction, distance, or position. When representing spatial direction it is convenient to approximate spatial entities by points. But this is inappropriate as far as spatial topological information is concerned: topology concerns sets of points, i.e. regions.

2.3.1 RCC-8 Topological Relations

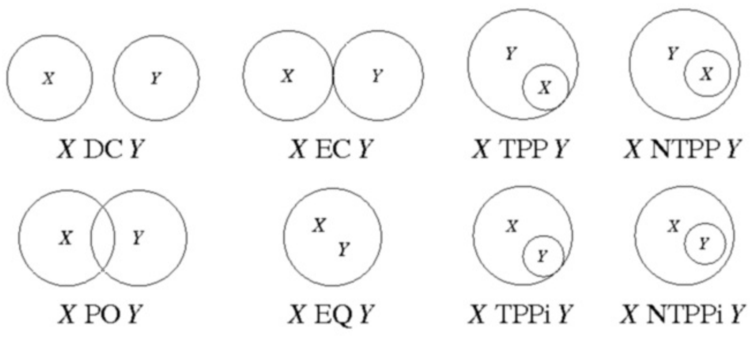


Figure 2.1: The set of RCC-8 Topologic Relations

The region connection calculus (RCC) serves for qualitative spatial representation and reasoning. RCC abstractly describes regions (in Euclidean space, or in a topological space) by their possible relations to each other. RCC8 (Figure 2.1) consists of 8 basic relations that are possible between two regions:

- disconnected (DC)
- externally connected (EC)
- equal (EQ)
- partially overlapping (PO)
- tangential proper part (TPP)
- tangential proper part inverse (TPPi)
- non-tangential proper part (NTPP)
- non-tangential proper part inverse (NTPPi)

2.3.2 Cone-shaped Directional Relations

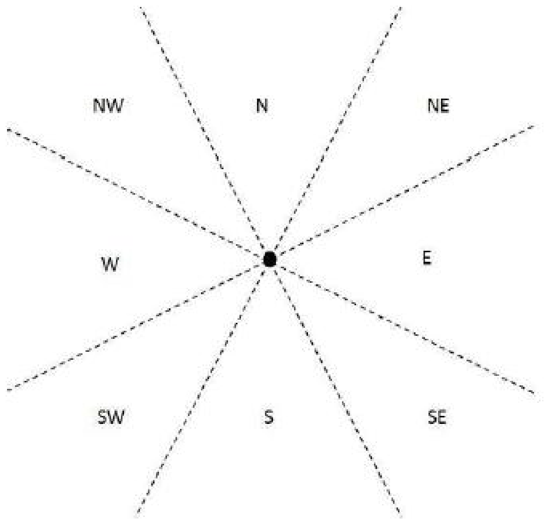


Figure 2.2: The set of Cone Shaped Directional Relations

The goal of a qualitative representation of the direction between points in two-dimensional space is to specify a limited number of relations such that each relation covers a part of the 360 degrees range and all relations taken together cover the 360 degrees range completely. If in addition the relations do not overlap, they form a jointly exhaustive and pairwise disjoint (JEPD) set of relations, called basic relations. In CSD model an angular direction is assigned the nearest named direction which results in cone-shaped areas for which a symbolic direction is applicable. This model has the property that "the area of acceptance for any given direction increases with distance" [23] and is sometimes called 'triangular'. Cone-shaped Directional (Figure 2.2) defines the set of 9 basic relations that are possible between two points (e.g., region's centroid), with 8 turns of 45 degrees being the identity function:

- north (N)
- north-east (NE)
- east (E)
- south-east (SE)
- south (S)
- south-west (SW)
- west (W)
- north-west (NW)
- identical (O)

One can also form a subset without 0 $\{N, NE, E, SE, S, SW, W, NW\}$.

2.4 SOWL

SOWL [7], is an ontology for representing and reasoning over spatio-temporal information in OWL. Building upon well established standards of the semantic web (OWL 2.0, SWRL) SOWL enables representation of static as well as of dynamic information based on the 4D-fluents (or, equivalently, on the N-ary) approach [25, 26] . Both RCC-8 toplogical and cone-shaped directional relations are integrated in SOWL. Representing both qualitative temporal and spatial information (i.e., information whose temporal or spatial extents are unknown such as "left-of" for spatial and "before" for temporal relations) in addition to quantitative information (i.e., where temporal and spatial information is defined precisely) is a distinctive feature of SOWL. The SOWL reasoner is capable of inferring new relations and checking their consistency, while retaining soundness, completeness, and tractability over the supported sets of relations.

The SOWL spatial representation implements reasoning rules for RCC-8 relations and cone-shaped direction relations using SWRL and OWL 2.0 property axioms. Specifically, the nine direction relations have been declared as transitive OWL relations (i.e., a relation such as South is transitive meaning that if the relation holds between locations A and B, and between locations B and C, it also holds between locations A and C). Their inverse relations (e.g., North is the inverse of South) are defined as well. Furthermore, the identity relation (O) is symmetric. All basic relations are pairwise disjoint. Path consistency is implemented by introducing rules defining compositions and intersections of supported relations until a fixed point is reached or until an inconsistency is detected. Reasoners that support DL-safe rules such as Pellet can be used for inference and consistency checking over spatio-temporal relations.

2.5 PelletSpatial

PelletSpatial [4] extends Pellet OWL reasoner with qualitative spatial reasoning capabilities. It supports checking the consistency of spatial relations expressed using RCC-8 and computes new spatial inferences from asserted relations. The spatial relations are expressed in RDF/OWL and can be combined with arbitrary domain ontologies. PelletSpatial can answer SPARQL queries that mix spatial relations with arbitrary RDF/OWL relations. PelletSpatial implements two RCC reasoners: (a) A reasoner based on the semantics preserving translation of RCC relations to OWL-DL class axioms and (b) a reasoner based on the RCC composition table that implements the path-consistency algorithm of 3.4.

Results show that, without further optimizations, the first reasoner based on the translation of RCC relations to OWL-DL class axioms lacks practicability even for small datasets. In addition to translation RCC relations to OWL class axioms, one axiom is defined for each region to satisfy the regularity condition of region (to be a non-empty concept and to contain all of the regions interior points). This axiom significantly affects non-determinism as well as the number of qualified existential quantifiers in the ontology. Qualified existential and universal quantifiers, is one of the source of complexity (AND-branching) in DL reasoning [17]. The second spatial reasoner based on a path-consistency algorithm and the RCC-8 composition table has been shown to be more promising with regards to performance.

2.6 SPARQL

The SPARQL query language [24] can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

2.7 Jena

Jena [27] is an open source Semantic Web framework for Java. It provides an API for reading as well as for writing to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A model can also be queried using SPARQL [24] and updated using SPARUL. Jena also works with OWL (Web Ontology Language). The framework has various internal reasoners such as Pellet [5, 6].

2.8 Protégé editor

Protégé [2] is a free, open-source platform that provides tools to construct domain models and knowledge-based applications with ontologies. It supports the creation, visualization and manipulation of ontologies in various representation formats. Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé platform supports two main ways of modeling ontologies: the Protégé-Frames editor and the Protégé-OWL editor. The Protégé-OWL editor enables users to build ontologies for the Semantic Web in the W3C's Web Ontology Language (OWL). "An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e., facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms" [1].

Protégé also supports the editing and execution of SWRL rules. It provides a set of libraries that can be used in rules, including libraries to interoperate with XML documents, and spreadsheets, and libraries with mathematical, string, RDFS, and temporal operators.

Chapter 3

CHOROS Reasoning

3.1 Handling Spatial Ontologies

Spatial relations are expressed in RDF/OWL and can be combined with standard RDF/OWL semantic relations forming an ontology. A relation is represented as a triple. The predicate will be one of the terms used to express a spatial relation, while the subject and object will be the regions or the points involved in the relation. In OWL, such a statement is called an "Object Property Assertion Axiom" (e.g., Individual:Region1 ObjectProperty:disconnectedFrom Individual:Region2).

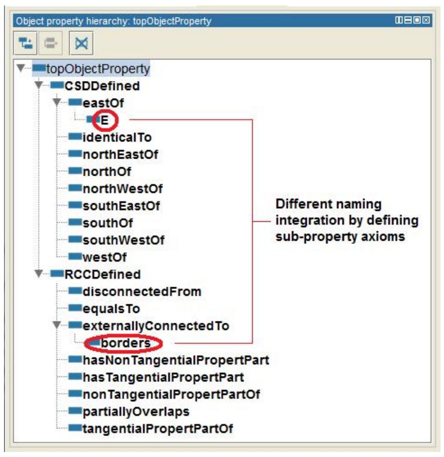


Figure 3.1: RCC-8 and CSD as object properties, in Protégé

CHOROS defines an RDF/OWL vocabulary for expressing qualitative spatial relations, with both the RCC-8 topological and Cone-shaped Directional models. Henceforth, we refer to these calculi as RCC and CSD respectively. As illustrated in Fig. 3.1, RCC and CSD terms are defined as simple object properties with no extra characteristics(e.g., inverse, transitive). One can either use the vocabulary provided, or use its own by defining sub-property axioms.

Katz et al. [8] propose representing RCC-8 as OWL-DL class axioms, but this approach does not scale-up well for many relations [4].

3.2 Reasoning Architecture

In CHOROS spatial knowledge is represented by an OWL ontology. Briefly,

- we represent every region as an OWL individual (e.g., Individual:Town1, Individual:Town2 of type Class:Town)
- we define an OWL object property for each spatial relation (see Fig. 3.1)
- a spatial relation between two regions is represented as an OWL object property assertion (e.g., Individual:Town1 ObjectProperty:westOf Individual:Town2).

Non-spatial relations, such as region type, size, etc., are represented as ordinary OWL assertions (e.g., Individual:Town1 DatatypeProperty:hasName Individual:Chania). This approach helps us to support reasoning and querying for both spatial relations and standard RDF semantic relations by setting apart each problem. Thus, CHOROS strictly separates spatial reasoning from semantic OWL-DL reasoning by using an exclusive spatial reasoner component (CT_Reasoner).

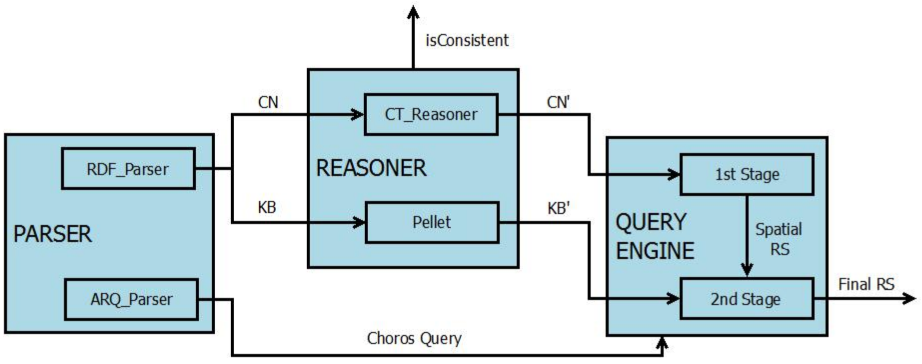


Figure 3.2: Main Components of the CHOROS reasoner

Fig. 3.2 illustrates the main components of the CHOROS reasoner. Ontologies are loaded into the Parser after a step of validation. This step ensures that all resources have a valid triple form. During the loading phase (RDF-Parsing), spatial property assertions are put into a constraint network (CN) and non-spatial standard OWL assertions are stored in a knowledge base (KB). The core of the system is the Reasoner component. The Composition Table Reasoner (CT_Reasoner) checks the consistency of a constraint network, while Pellet checks the consistency of the knowledge base.

The queries are also loaded after validation. During the loading phase (ARQParsing), a query structure (Choros Query) is created in order to compartmentalize query atoms into spatial and non-spatial. The Query Engine component applies a dual stage query answering technique. The First Stage returns a set of spatial query results (spatial RS). This set is given as input to the Second Stage, consisting of further constraining such that the non-spatial query is satisfied. Thus, we get the final set of query results (final RS).

We described the main idea of reasoning architecture of PelletSpatial, that we also followed in CHOROS. In the next three subsections we describe our implementation and its extensions in detail. But for a closer look, we must first give a short explanation of some commonly used terms:

- A Knowledge base (KB) is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. The Pellet KB is a combination of an assertional box (component that contains assertions about individuals) and a terminological box (component that contains axioms about classes), which provides consistency checking and query services.
- A constraint network (CN) is a set of variables together with a set of constraints and perhaps also one or more objective functions. Spatial constraint networks allows to store and handle spatial knowledge by providing similar to a KB functionality for checking consistency as well as for querying.

3.2.1 Parser

The Parser component is composed of a RDF parser for loading ontologies and an ARQ parser for loading queries.

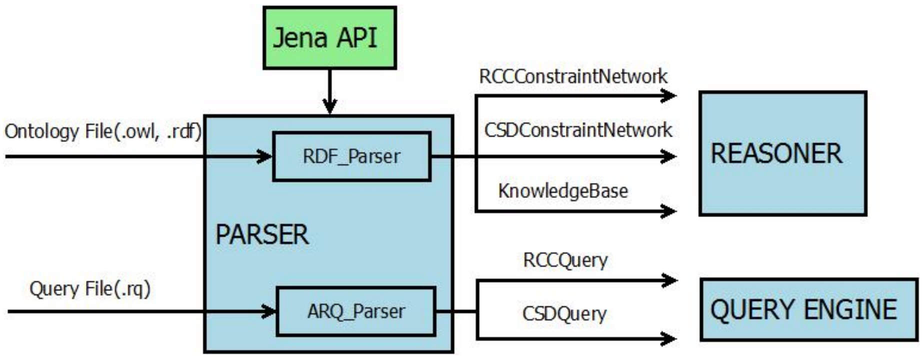


Figure 3.3: Parser Component

In RDF parsing, we use Jena API (2.7) and internal spatial vocabularies of the Parser component to extract data from the RDF graph. We have created theses vocabularies (as Jena model properties) to define the basic relations of RCC-8 and CSD formalisms. Respectively, we create two constraint networks, one for each of the above spatial calculus. Thus, we can track and remove any spatial triple from the graph, in order to store it in the corresponding constraint network. The rest of the graph will be managed as a Pellet KB.

In ARQ parsing, we use Jena API and internal query formats (RCCQuery, CSDQuery) in order to divide query atoms into topological RCC-8, cone-shaped directional and non-spatial. The same spatial vocabularies are used to map triple predicates to spatial relations.

3.2.2 Reasoner

The core of the system is the Reasoner component. CHOROS strictly separates spatial reasoning from semantic OWL-DL reasoning, as it uses one exclusive reasoner for each calculus: RCC relations are managed as a RCC constraint network, CSD relations are managed by a CSD constraint network as well. Non-spatial relations are managed by Pellet as a KB.

In both CSD and RCC-8 qualitative formalisms, relations are expressed based on a set of jointly exhaustive and pairwise disjoint basic relations which is closed under several operations. Thus, it is possible to apply constraint based methods for reasoning over these relations. For this, it is necessary to give a composition table either for all relations, or for the basic relations only together with procedures for computing the compositions of complex relations. A composition table is defined using the formal semantics of the relations. Otherwise it is not possible to verify correctness and completeness of the inferences. Formal semantics of the relations are also necessary for finding efficient reasoning algorithms which are essential for most applications. Without formal semantics it is sometimes not even possible to show that reasoning over a system of relations is decidable.

	DC	**EC**	**PO**	**TPP**	**NTPP**	**TPPi**	**NTPPi**	**EQ**
DC	DC, EC, PO, TPP, NTPP, TPPi, NTPPi, EQ	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, 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	DC, EC, PO, TPP, NTPP, TPPi, NTPPi, EQ	PO, TPP, NTPP	PO, TPP, NTPP	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPPi, NTPPi	PO
TPP	DC	DC, EC	DC, EC, PO, TPP, NTPP	TPP, NTPP	NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPPi, NTPPi	TPP
NTPP	DC	DC	DC, EC, PO, TPP, NTPP	NTPP	NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP, TPPi, NTPPi, EQ	NTPP
TPPi	DC, EC, PO, TPPi, NTPPi	EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	EQ, PO, TPPi, TPP	PO, TPP, NTPP	TPPi, NTPPi	NTPPi	TPPi
NTPPi	DC, EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPP, NTPP, EQ, TPPi, NTPPi	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

Table 3.1: RCC Composition Table

	N	**NE**	**E**	**SE**	**S**	**SW**	**W**	**NW**	**O**
N	N	N,NE	N,NE,E	N, NE, E, SE	N, NE, E, SE, S, SW, W, NW, O	W, NW, SW, N	NW, W, N	NW,W	N
NE	NE, E	NE	NE, E	E, NE, SE	E, NE, SE, S	N, NE, E, SE, S, SW, W, NW, O	N, NE, NW, W	N, NE, NW	NE
E	NE, E, N	NE, E	E	SE, E	SE, E, S	S, SW, SE, E	N, NE, E, SE, S, SW, W, NW, O	N, NW, NE, E	E
SE	E, SE, NE, N	E, SE, NE	SE, E	SE	SE, S	S, SE, SW	S, SE, SW	N, NE, E, SE, S, SW, W, NW, O	SE
S	N, NE, E, SE, S, SW, W, NW, O	E, S, NE, SE	SE, E, S	SE, S	S	S, SW	S, W, SW	W, S, NW, SW	S
SW	W, SW, N, NW	N, NE, E, SE, S, SW, W, NW, O	S, SW, SE, E	S, SW, SE	SW,S	SW	SW, W	W, NW, SW	SW
W	N, W, NW	N, NW, NE, W	N, NE, E, SE, S, SW, W, NW, O	S, SE, SW, W	W, S, SW	W, SW	W	W, NW	W
NW	N, NW	N, NW, NE	N, NW, NE, E	N, NE, E, SE, S, SW, W	W, NW, SW, S	W, NW, SW	NW, W	NW	NW
O	N	NE	E	SE	S	SW	W	NW	O

Table 3.2: CSD Composition Table

In our implementation, consistency checking for each constraint network is performed by means of a path-consistency algorithm (see 3.2.2). This algorithm is based on the corresponding composition table for RCC-8 (see Table 3.1) and CSD (see Table 3.2) relations respectively. Currently, CHOROS supports path-consistency based reasoning only for the basic relations of each calculus as they are defined in SOWL [7]. For a closer look consider the following examples:

Consistent Example

We have four houses. The first house is north of the second and northeast of the fourth; the second house is northwest of the third; the fourth house is north of the third. What can we infer about the relation between the first and the third house?

The spatial configuration can be formalized in CSD as the following constraint network:

- house1 N house2
- house2 NW house3
- house1 NE house4
- house4 N house3

Using the CSD composition table 3.2 and the path-consistency algorithm described in 3.2.2, we can refine the network in the following way:

- house1 N, NW house3
- house1 N, NE house3

That is, the first house is north of the third which is the intersection of the above two relations.

Inconsistent Example

Now let the fourth house be east of the third. The spatial configuration can be formalized in CSD as the following constraint network:

- house1 N house2
- house2 NW house3
- house1 NE house4
- house4 E house3

Using the CSD composition table and the path-consistency algorithm, we can refine the network in the following way:

- house1 N, NW house3
- house1 E, NE house3

That is, network is inconsistent because the intersection of the above two relations is the empty relation.

Path-Consistency Algorithm

In this section, we describe the algorithm (Figure 3.4) used for checking the consistency of a constraint network . It is the same implementation used in PelletSpatial [4] in the hybrid architecture.

Given a constraint network N, i.e., a set of defined RCC-8 relations, N is consistent if it is either empty, or if every relation in the network is consistent. Notice that, the requirement for the relations in N to be defined, i.e., of the set of eight base relations, is relevant to the tractability of a sound and complete path-consistency procedure. As it is noticed in [9, 7], for relations defined over RCC-8 or CSD, path consistency is tractable.

The complete step (Line 5) processes N to infer all the inverse and equals relations. For every (defined) relation $R_{ij} \in$ N, we ensure that $R_{ji}^{\breve{}} \in$ N (inverse complete), e.g., for TPP(a, b) we ensure that TPPi(b, a) $\in$ N, a,b regions $\in$ N. For every region a $\in$ N, we ensure that EQ(a, a) $\in$ N (equals complete). A queue Q is used as a structure to keep track of relations that have to be processed. Hence, the algorithm runs until $Q = \emptyset$ or an inconsistency is detected. Q is initialized with all the defined relations $R_{ij} \in$ N.

```
 1: procedure PathConsistency($N$)
 2:     if $N = \emptyset$ then
 3:         return true
 4:     end if
 5:     $complete(N)$
 6:     $Q \leftarrow \{R_{ij} | R_{ij} \in N\}$
 7:     while $Q \neq \emptyset$ do
 8:         $R_{ab} \leftarrow remove(Q)$
 9:         if $!isConsistent(N, Q, R_{ab})$ then
10:             return false
11:         end if
12:     end while
13:     return true
14: end procedure

15: procedure IsConsistent($N, Q, R_{ab}$)
16:     for $S_{bc} \in N$ do
17:         $T_{ac} \leftarrow R_{ab} \circ S_{bc}$
18:         $add(N, Q, T_{ac})$
19:         if $!isConsistent$ then
20:             return false
21:         end if
22:     end for
23:     return true
24: end procedure

25: procedure Add($N, Q, T_{ac}$)
26:     if $T = \top$ then
27:         return
28:     end if
29:     $U_{ac} \leftarrow \{R_{ij} | i = a, j = c, R_{ij} \in N\}$
30:     if $\nexists U_{ac}$ then
31:         $V_{ac} \leftarrow T_{ac}$
32:     else
33:         $V_{ac} \leftarrow T_{ac} \cap U_{ac}$
34:         if $V = \emptyset$ then
35:             $isConsistent = false$
36:             return
37:         end if
38:         if $U = V$ then
39:             return
40:         end if
41:         $N \leftarrow N \setminus \{U_{ac}\}$
42:     end if
43:     $N \leftarrow N \cup \{V_{ac}\}$
44:     $Q \leftarrow Q \cup \{V_{ac}\}$
45:     $add(N, Q, V_{ca}^{\breve{}})$
46: end procedure
```

Figure 3.4: Path Consistency Algorithm

A relation R_{ab} (Line 15) is path-consistent if the rule for combining a compositional inference with existing information [10],

$$V_{ac} \leftarrow U_{ac} \cap R_{ab} \circ S_{bc}$$

results in a non-empty set $V \neq \emptyset$ for regions a, c; $S_{bc} \in N$ relations with a transitive path with R_{ab} from a through b to c and U_{ac} a relation (possibly $\in N$ as existing information).

The compositional inference $T_{ac} \leftarrow R_{ab} \circ S_{bc}$ (Line 17) is computed for regions a, c as the union set T for the composition of each pair (r, s) in the set $R \times S, r \in R, s \in S$. The composition of a pair (r, s) consists in a lookup for the RCC-8 (or CSD) composition table given that r and s are elements of the set of eight defined relations.

If $U_{ac} \in N$, i.e., there is existing information for the pair (a, c), we complete the rule by computing the intersection $V_{ac} \leftarrow T_{ac} \cap U_{ac}$ (Line 33), where V is the intersection set of relations $v \in T \cap U$. This step does refine the already existing relation $U_{ac} \in N$ and is essential for the path-consistency algorithm as it defines the inconsistent state: if $V = \emptyset$. we have found an inconsistency.

The step of Line 38 states that, if U = V the step at Line 33 could not refine relation U_{ac}. Hence, combining compositional inference T_{ac} with existing information U_{ac} does not add new information. In this case, we can return. Else, U_{ac} is removed from N, the refined V_{ac} is added to N and Q and the inverse $V\breve{}_{ca}$ is processed.

Standard path-consistency algorithms, like the implementation described in [9], usually use a $n \times n$ dense matrix M for n different regions, where M_{ij} represents the relation between the regions i, j. Instead, the implementation in CHOROS processes a sparse matrix with empty cells for $M_{ij} = \top$. Thus, Q corresponds to the array of elements $M_{ij} \neq \top$. Furthermore, a triple structure allows to be more compact in the rule for combining compositional inference and existing information, as the triple already contains all the required information to compute the rule. Finally, we split the compositional inference from its combination with existing information. This allows us to discard the compositional inference whenever it returns the universal relation T from further processing as it cannot add any new information to the network.

3.2.3 Query Engine

As CHOROS processes both spatial and standard semantic OWL relations in RDF/OWL documents, it is natural to support spatial querying. Query Engine component answers conjunctive queries that include spatial and non-spatial patterns, i.e., triple patterns for spatial relations (joined) with triple patterns for semantic RDF relations. More specifically, this module supports a subset of queries written in SPARQL, that satisfies the following conditions:

- No variable is used in the predicate position.
- Each property used in the predicate position is either a property (object or datatype) defined in the ontology or one of the following built-in properties: rdf:type, owl:sameIndividualAs, owl:differentFrom.
- At least one of the triples must contain a ”spatial” object property in the predicate position.

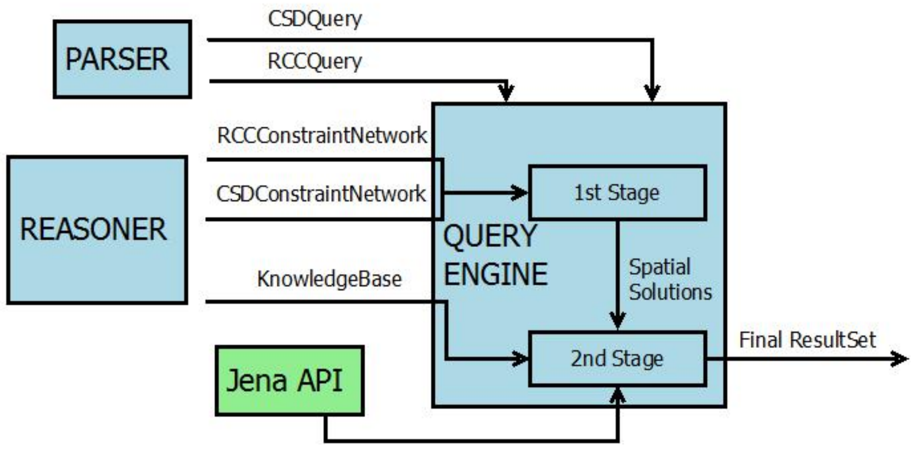


Figure 3.5: Query Engine Component

The CHOROS Query Engine uses a dual stage query answering technique by extending each constraint network with a dedicated query handler that, given a spatial query atom and (for conjunctive queries) a set of prepared bindings, returns a set of spatial query solutions. The set of query solutions returned by the first stage is given as input to the second stage which consists of further constraining the set of bindings such that the non-spatial query subset is satisfied. If the first stage does not return any solution, second stage is ignored.

It is possible to query for regions that are involved in a specific spatial relation with another region and have certain characteristics that are described by semantic RDF relations. It is also possible to query exclusively for spatial patterns, but it is not possible to query for non-spatial patterns alone. For example, we can query a hypothetical ontology for all region names that are ”externally connected to” and ”north of” a given region (the spatial subset query) that have an area and population greater than a given lower bound (the non-spatial subset query).

3.3 Optimazations

In this section we present and discuss different versions of CHOROS along with possible optimizations.

CSD Calculus Approaches

The first version of CHOROS (Choros 0.1) applies to consistency checking over a CSD calculus of 9 basic relations. This implementation has poor performance for small datasets as it is very wasteful to pre-compute the compositions of 9 relations (512x512) and store them in the full composition table. CHOROS version 0.2 applies to consistency checking over a CSD calculus of 8 basic relations. Spatial relation "identical to" is replaced by the owl axiom "sameAs". Thus, we have to pre-compute and store less data (256x256), by computing the compositions of 8 relations instead of 9, which is much faster.

Multithreading

Multithreading allows two parts of the same program to run concurrently. Code parallelization is the process of modifying a simulation code to make it run faster by splitting the workload among multiple computers (well, in the very general sense). Parallelization of a serial code is a nontrivial task. It requires significant code changes and debugging. If the number of required cases is larger than the number of CPUs, such an approach will result in non-optimal performance. We utilize multithreading by launching each case as a separate thread. A simple scheduler ensures that the number of threads running at any instance of time equals the number of CPU cores (2 in our case). And while OpenMP [28] is employed to add multithreading into C codes, Java supports multithreading natively. In CHOROS, which is implemented in Java, multithreading enables "parallel" execution of CSD and RCC consistency checking, as we launch each task as a separate thread.

Chapter 4

Experiments

4.1 Complexity

The complexity of a decision problem is usually measured in term of the worst-case running time or memory consumption. Running time as well as memory consumption of an algorithm depends on the size n of its input, i.e., the size of the problem instance, and can be expressed as a function f(n). The asymptotic behavior of f is specified in terms of the O-notation which gives an upper bound on the running time. In areas like database systems where instances are very large size, a running time of $O(n^3)$ corresponds to a solution which is very slow in practice. In these cases efficient algorithms have linear running time.

For most of the tractable subsets of qualitative spatial calculi, path-consistency or even simpler methods are sufficient for deciding consistency. So except for very large instances or for calculi over a large set of relations, there are usually no efficiency problems.As shown in [11], by restricting the supported relations set to a tractable subset of RCC-8 (or corresponding directional) spatial relations, path consistency has $O(n^5)$ time complexity (with n being the number of individuals) and is sound and complete. Note that, extending the model for the full set of relations would result into an intractable reasoning procedure.

In our implementation, as well as in [4], path consistency has $O(n^3)$ worst time complexity. In the following we will show results from an empirical investigation of the practical efficiency of CHOROS. To assess the performances of CHOROS in practice, we created an example ontology which was subsequently populated with random instances. This ontology is described in detail in the next section.

4.2 Case Study - TUC Spatial Ontology

The ”TUC spatial ontology” is a mechanism to describe data related to spatial entities of the University campus of Technical University of Crete (TUC).

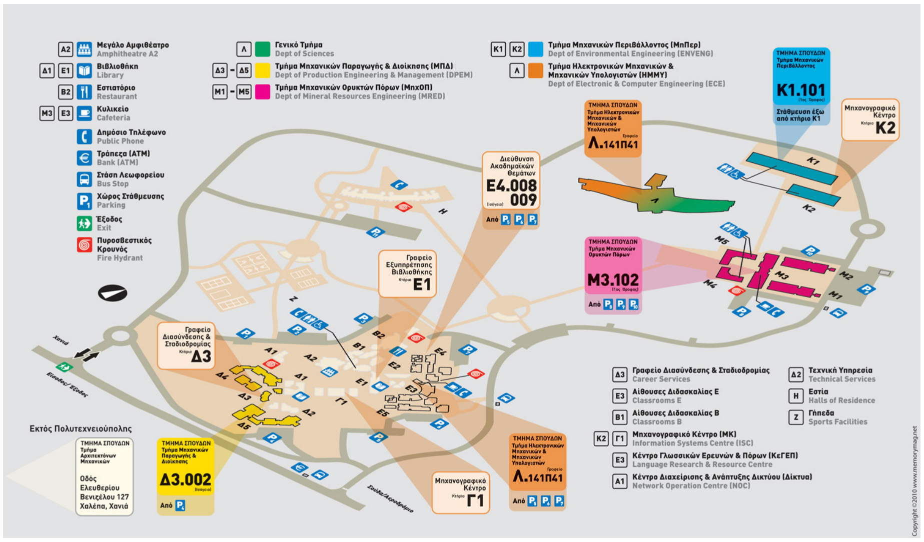

Μεγάλο Αμφιθέατρο
Amphitheatre A2
Βιβλιοθήκη
Library
Εστιατόριο
Restaurant
Κυλικείο
Cafeteria
Δημόσιο Τηλέφωνο
Public Phone
Τράπεζα (ATM)
Bank (ATM)
Στάση Λεωφορείου
Bus Stop
Χώρος Στάθμευσης
Parking
Έξοδος
Exit
Πυροσβεστικός Κρουνός
Fire Hydrant
Γενικό Τμήμα
Dept of Sciences
Τμήμα Μηχανικών Παραγωγής & Διοίκησης (ΜΠΔ)
Dept of Production Engineering & Management (DPEM)
Τμήμα Μηχανικών Ορυκτών Πόρων (ΜηχΟΠ)
Dept of Mineral Resources Engineering (MRED)
Τμήμα Μηχανικών Περιβάλλοντος (ΜηΠερ)
Dept of Environmental Engineering (ENVENG)
Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών (ΗΜΜΥ)
Dept of Electronic & Computer Engineering (ECE)
K1.101
K2
Λ.141Π41
E4.008 009
E1
M3.102
Δ3
Δ3.002
Γ1
Εκτός Πολυτεχνειούπολης
Γραφείο Διασύνδεσης & Σταδιοδρομίας
Career Services
Αίθουσες Διδασκαλίας Ε
Classrooms E
Αίθουσες Διδασκαλίας Β
Classrooms B
Μηχανογραφικό Κέντρο (ΜΚ)
Information Systems Centre (ISC)
Κέντρο Γλωσσικών Ερευνών & Πόρων (ΚεΓΕΠ)
Language Research & Resource Centre
Κέντρο Διαχείρισης & Ανάπτυξης Δικτύου (Δίκτυα)
Network Operation Centre (NOC)
Τεχνική Υπηρεσία
Technical Services
Εστία
Halls of Residence
Γήπεδα
Sports Facilities

Figure 4.1: Campus Map - Technical University of Crete

Figure 4.1 illustrates the campus map of TUC. Information in this map is used to acquire the necessary concepts for defining a basic hierarchical structure. However, it was not sufficient for discriminating the relative position of regions. A Google’s satellite image, Figure 4.2, gave us a panoramic view of the area and made it possible to define qualitative spatial relations between regions.

The ”TUC spatial ontology” is implemented in OWL. It consists of classes, properties and individuals. Table 4.1 shows all classes, gives a brief description and lists some individuals that belong to each class. Table 4.2 and Table 4.3 illustrate object properties and datatype properties.

We used the Google’s satellite image to define qualitative spatial relations between regions, especially cone-shaped directional relations between their centroids. We located the individuals representing regions of the campus, according to the class they belong. We created one picture for almost every class of our ontology. Sub-figures of 4.2 depict the classified regions. The colored square symbols represent the centroids of these regions. Adobe Photoshop C5 [29] proved very helpful for this. So for example, the ”Ruler Tool” was used to position elements precisely and measure the angles between them.

Class			Description	Individual
Region			Superclass of the ontology. Every individual of this ontology represents a region. Individuals not belonging in any of the other subclasses are defined here directly.	Campus,the road inside the campus and the gate.
	Bus Stop		Bus stops serving a number of destinations throughout the campus.	Bus stop serving ECE department and 5 more bs.
	Classrooms		Groups of classrooms or classrooms-buildings. Parts of a classroom group should not be defined as an individual of this class.	Amphitheatre, group E and 4 more groups.
	Department		Departments of the Technical University of Crete.	ECE, DPEM and 4 more departments.
	Facilities		Facilities of the campus.	
		Sports	Facilities of the campus for sports activities.	Basketball courts and 4 more courts.
		Residence	Facilities of the campus for student's accommodation.	Hestia.
		Food	Facilities of the campus for food serving.	Restaurant, 3 Cafeterias.
	Parking		Areas that serve parking throughout the campus.	Parking area 1-14.
	Services		Central offices, libraries and other buildings providing services.	NOC, ISC 1-2, Career Services and 5 more.

Table 4.1: Classes and instances in the TUC spatial ontology.

Object Property		**Description**	**Subject class to be applied (domain)**	**Object class to be applied (range)**
CSDDefined		Cone shaped directional relations between centroids of regions.	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.
	northOf			
	northEastOf			
	eastOf			
	southEastOf			
	southOf			
	southWestOf			
	westOf			
	northWestOf			
	identicalTo			
RCCDefined		RCC-8 topological relations between regions.	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.
	disconnectedFrom			
	equalsTo			
	externallyConnectedTo			
	hasNonTangentialProperPart			
	hasTangentialProperPart			
	nonTangentialProperPartOf			
	partiallyOverlaps			
	tangentialProperPartOf			
hasParking		Non-spatial relation representing that a region is served by a parking area.	Classrooms, Department, Facilities and Services.	Parking.

Table 4.2: Object Properties in the TUC spatial ontology.

Datatype Property	**Description**	**Subject class to be applied (domain)**	**Object class to be applied(range)**
hasName	Relation defining the full name of a region	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.	Type “String”
hasCodeName	Relation defining a code name of a region according to campus Map	Region, Bus Stop, Classrooms, Department, Facilities, Parking Services.	Type “String”

Table 4.3: Data Properties in the TUC spatial ontology

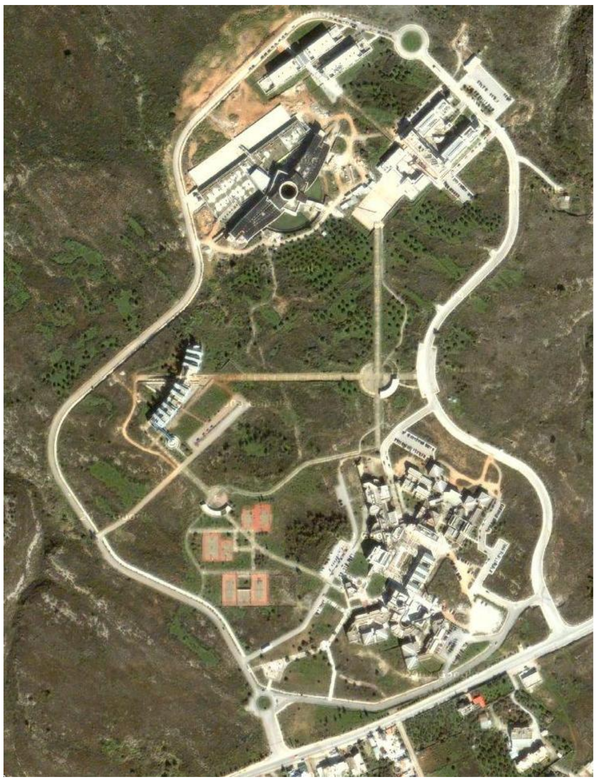
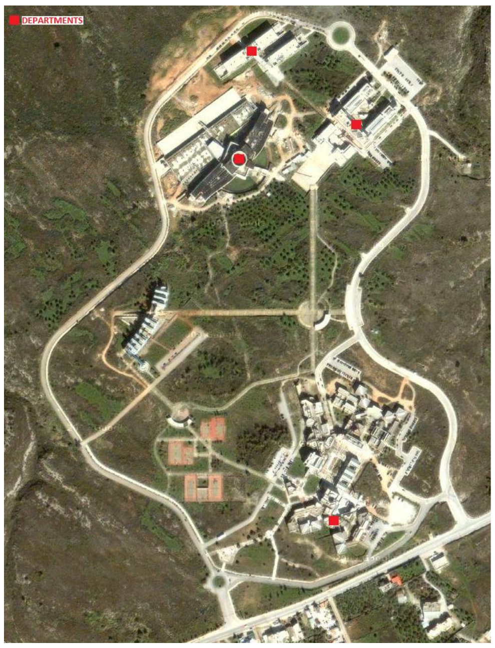
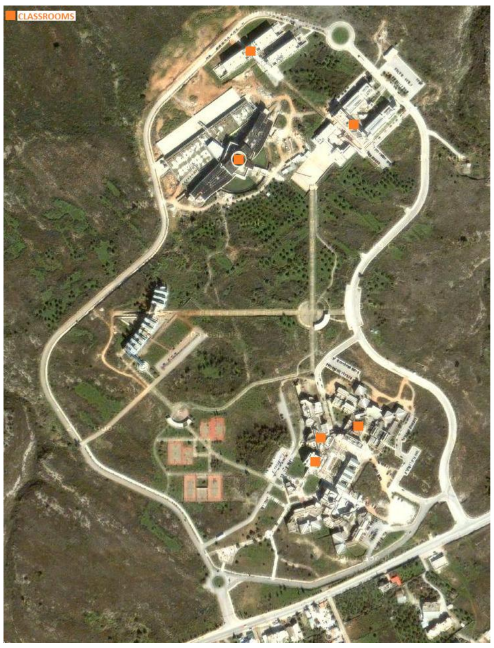


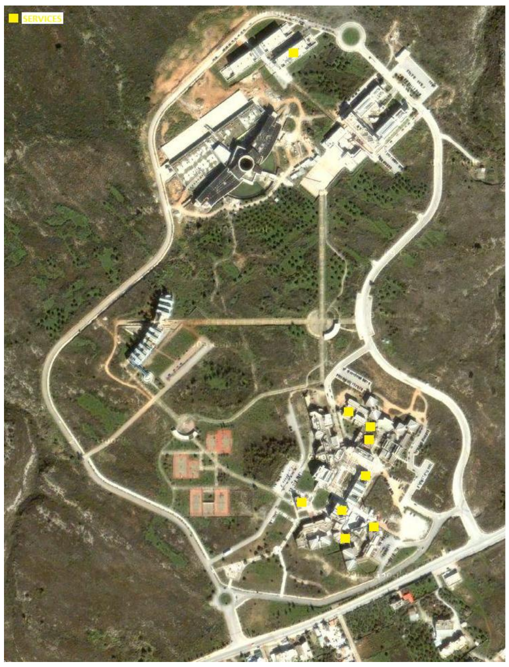
Figure 4.2: Google Satellite - Technical University of Crete



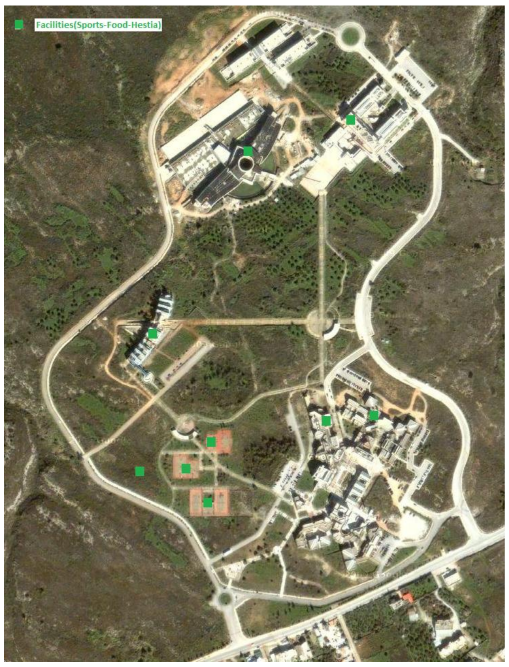
Departments of TUC



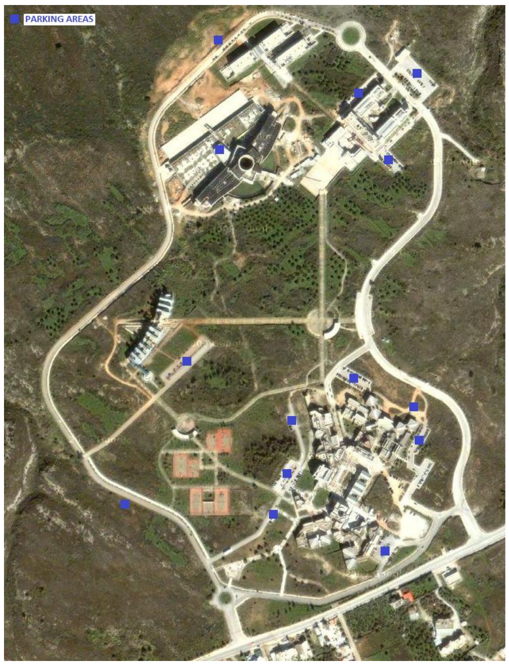
Groups of classrooms of TUC



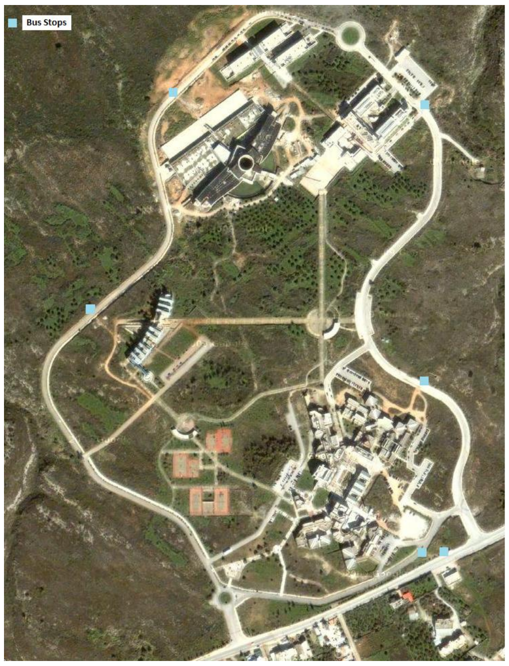
Services of TUC



Facilities(Sports-Food-Hestia) of TUC



Parkings of TUC



BusStops of TUC

4.3 Response Time

In order to test the practical efficiency of reasoning algorithms, it is necessary to generate a large number of test instances. Ideally, these must be real instances of existing applications. The "TUC spatial ontology" described in 4.2 it's such an application. However, the relatively small number of instances (regions) of the TUC ontology (in the order of 50) was a limitation for the thorough analysis of the performance of CHOROS. To objectively assess the performances of CHOROS, additional instances are generated using a random generator (100 instances into the TUC ontology). In the following, to demonstrate the performance of our reasoner, we discuss measurements of response time for reasoning as a function of the number of instances (individuals) in the TUC ontology. We separate these diagrams in two categories:

- Average case diagrams. Average measure of time the algorithm takes on a random input of n individuals (range 10-100). In real applications, much less than $O(n^2)$ relations (between n individuals) are asserted.
- Worst case diagrams. Longest running time performed by the algorithm given a hard input of n individuals (range 10-100). For n individuals, $O(n^2)$ relations are asserted into the knowledge base.

For each category we measured the performance of of the following implementations:

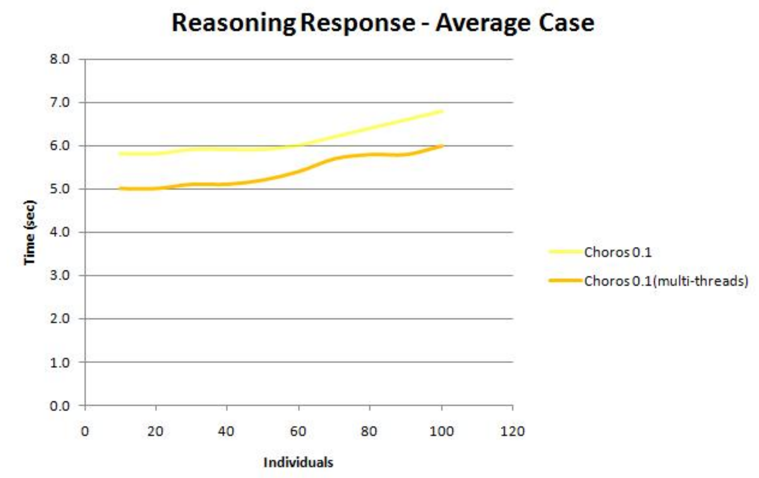
- Choros version 0.1 applies to consistency checking over a CSD calculus of 9 basic relations.
- Choros version 0.2 applies to consistency checking over a CSD calculus of 8 basic relations. Spatial relation "identical to" is replaced by the owl axiom "sameAs".
- Multithreading technique enables "parallel" execution of CSD and RCC consistency checking.

Figure 4.3 (a) shows average case complexity of Choros 0.1, contrasted with the multithreading technique implemented on the same version. Figure 4.3 (b) shows the same contrast for Choros 0.2, while in figure 4.3 (c) we compare the multithreaded approach of the two versions. Figure 4.4 illustrates results of worst case complexity, likewise.

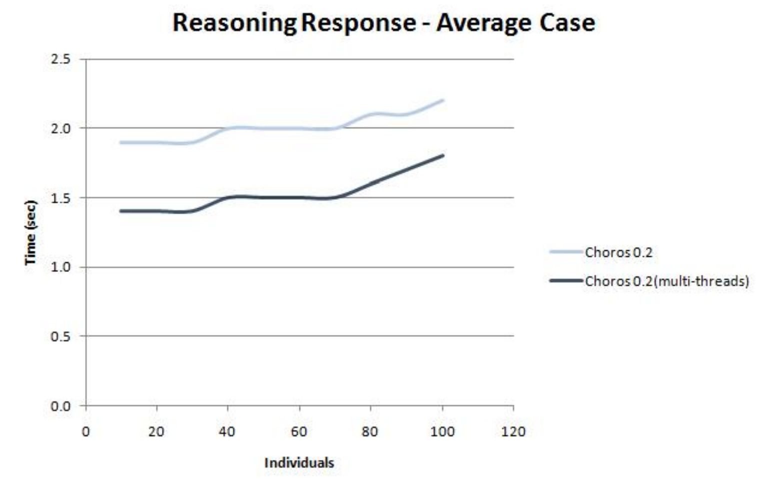
Computational complexity usually focuses on worst case complexity, but average case complexity is also important. In the following, the results confirmed our theoretical expectations.

The average case diagrams of Fig 4.3 reveal that , path-consistency algorithm in CHOROS exhibits nearly linear running time. On the other hand, worst case diagrams show that the algorithm is upper bounded by a polynomial expression in the size of its input. Specifically, in our implementation, path consistency has $O(n^3)$ worst time complexity.

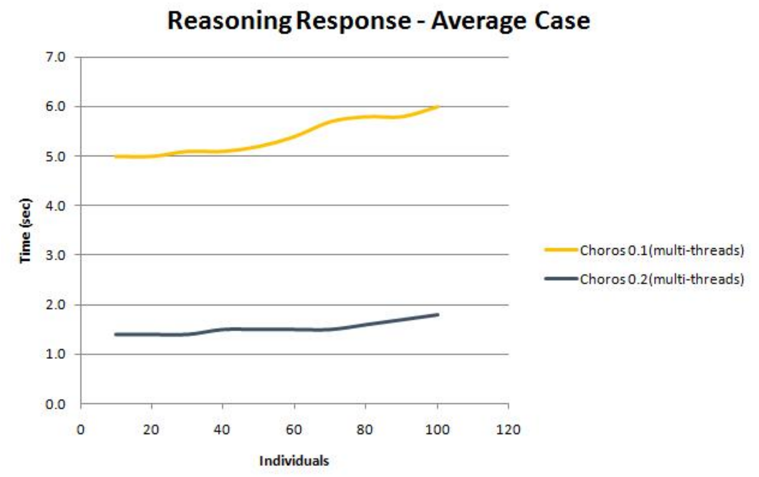
In the next section we compare our implementation of qualitative spatial reasoning of CHOROS with the SOWL reasoner implemented in SWRL [7]. The order of growth of the average-case and worst-case complexity is used to compare the efficiency of these two mechanisms.



(a) Choros 0.1

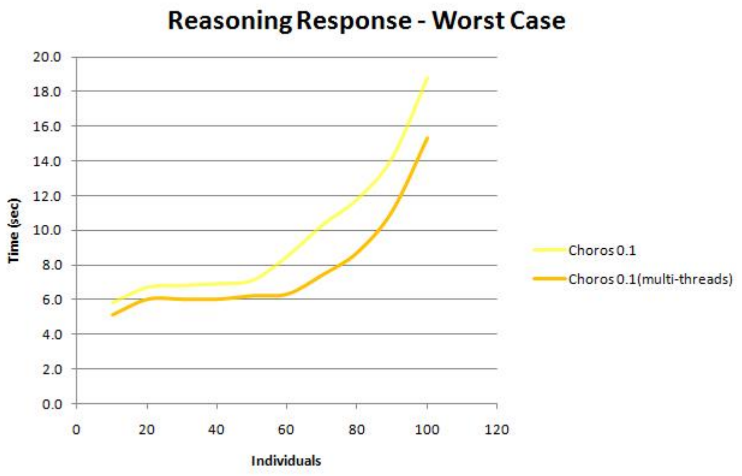


(b) Choros 0.2

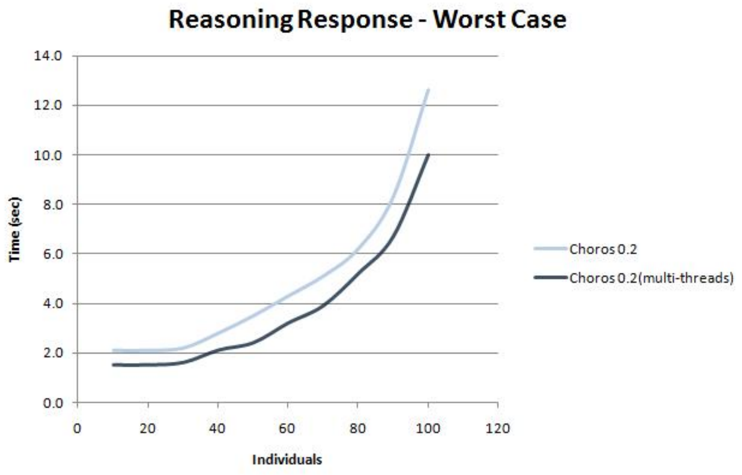


(c) Choros 0.1 vs Choros 0.2

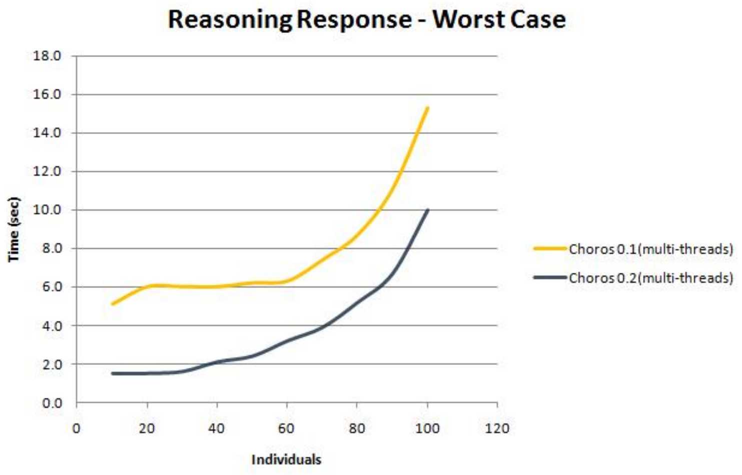
Figure 4.3: Average case diagrams of (a), (b) and (c)



(a) Choros 0.1



(b) Choros 0.2



(c) Choros 0.1 vs Choros 0.2

Figure 4.4: Worst case diagrams of (a), (b) and (c)

4.4 Implementing path consistency using Java compared to SWRL rules

Choros as well as SOWL support reasoning over qualitative spatial data. In this section we focus on the differences of these implementations, in order to distinguish the advantages and disadvantages of each.

	SOWL	**CHOROS**
Spatial Representation	RCC-8 relations and cone-shaped direction relations are defined as object properties with extra characteristics, i.e. disconnectedFrom, externallyConnected and partiallyOverlaps are declared as symmetric.	RCC-8 relations and cone-shaped direction relations are defined as simple object properties with no extra characteristics, i.e you don't have to declare in the ontology that, "tangentialProperPartOf" is the inverse property of "hasTangentialProperPart" or that "disconnectedFrom" is symmetric.
Reasoning Architecture	Path consistency is implemented by introducing SWRL rules defining compositions and intersections of supported relations until a fixed point is reached or until an inconsistency is detected. Pellet that supports DL-safe rules is used for inference and consistency checking over spatial and non spatial relations.	Consistency checking implementation strictly separates spatial reasoning from semantic OWL-DL reasoning by using one exclusive reasoner for each calculus. Pellet is used only for inference and consistency checking over non spatial relations.
Response Time	According to average case diagram (see Figure 4.5), consistency checking exhibits nearly linear running time. Path consistency has $O(n^3)$ worst time complexity (see Figure 4.6). Consistency checking over our case study (TUC spatial ontology) takes 2,3 seconds.	According to average case diagrams (see Figure 4.5), consistency checking exhibits nearly linear running time or even better. Path consistency has $O(n^3)$ worst time complexity (see Figure 4.6). Consistency checking of "TUC spatial ontology" takes 1,5 seconds.

Table 4.4: SOWL vs CHOROS

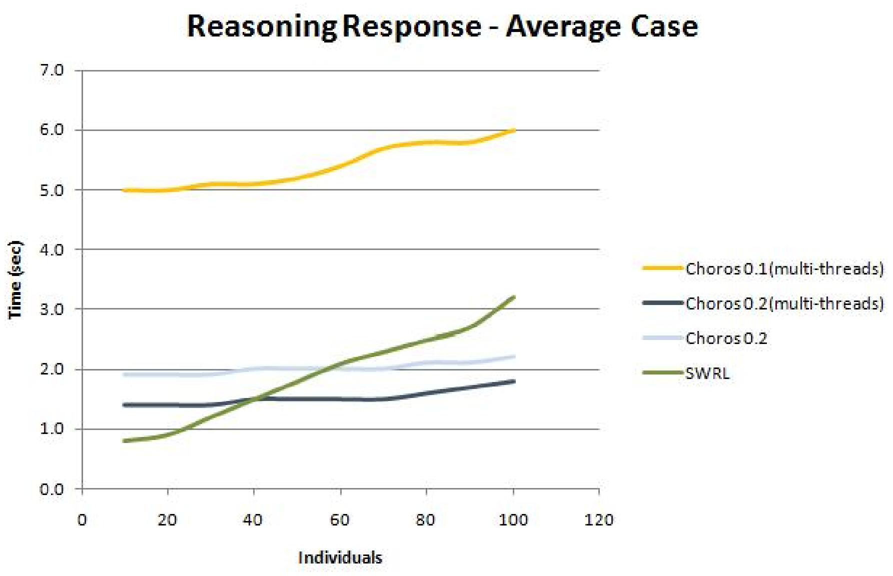


Figure 4.5: Average case - Choros vs SOWL

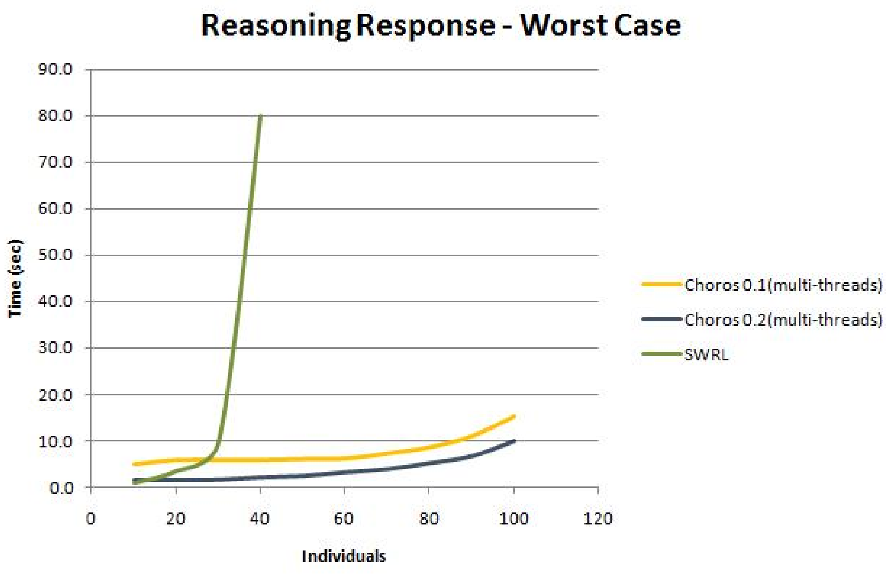


Figure 4.6: Worst case - Choros vs SOWL

Chapter 5

Conclusion and Future Work

CHOROS, is a qualitative spatial reasoning engine implemented in Java. CHOROS provides consistency checking and query answering over spatial data represented with the Region Connection Calculus (RCC) and the Cone-Shaped directional logic formalism (CSD). It supports all RCC-8 and CSD-9 relations as well as standard RDF/OWL semantic relations, both represented in RDF/OWL. As such, it can answer mixed SPARQL queries over spatial and non-spatial relation types. CHOROS extends PelletSpatial's hybrid architecture, which is based on a composition table that implements a path-consistency algorithm. We also, introduced a multithreading technique that enables to execute CSD and RCC consistency checking concurrently. As a case study we presented "TUC spatial ontology". Empirical investigation of the practical efficiency of our engine showed that consistency checking exhibits nearly linear running time averagely, and has $O(n^3)$ worst time complexity.

There are a number of directions for future work. We are planning to extend CHOROS support qualitative temporal reasoning on basic Allen relations. We can also support reasoning beyond the base relations of each calculi. Most tools support disjunctive relations between two regions to be asserted and a maximal tractable set of disjunctive relations has been shown to ensure the soundness and the completeness of the path-consistency algorithm. Scalability issues for large scale applications are also important issues for future research.

Bibliography

[1] D.L. McGuinness and F. VanHarmelen. OWL Web Ontology Language Overview", W3C Recommendation, February 2004. http://www.w3.org/TR/owl-guide/

[2] http://protege.stanford.edu/

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2nd Edition, 2007.

[4] M. Stocker, and E. Sirin "PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine" In: CEUR Workshop Proceedings, vol. 529-OWLED 2009, pp. 2-31, 2009.

[5] B. Parsia and E. Sirin. Pellet: An OWL DL Reasoner. In Proc. of Int. Workshop on Description Logics (DL2004), 2004.

[6] http://clarkparsia.com/pellet/

[7] Batsakis S., Petrakis E.: SOWL: A Framework for Handling Spatio-Temporal Information in OWL 2.0, 5th International Symposium on Rules: Research Based and Industry Focused (RuleML' 2011), Barcelona, Spain, July 19-21, 2011 (to appear).

[8] Y. Katz, and B. Grau. "Representing Qualitative Spatial Information in OWL-DL". In Proc. of Int.Workshop: OWL Experiences and Directions, Galway, Ireland, 2005.

[9] J. Renz and B. Nebel. Efficient Methods for Qualitative Spatial Reasoning. In Proc. of the 13th European Conference on Artificial Intelligence (ECAI98), 1998.

[10] B. Bennet. Knowledge Representation and Reasoning: Compositional Reasoning, 2007. Lecture notes, School of Computing, University of Leeds.

[11] J. Renz, and B. Nebel. Qualitative Spatial Reasoning using Constraint Calculi. In Handbook of Spatial Logics, Springer, Netherlands , pp. 161-215, 2007.

[12] B. Nebel, and H.J. Burckert. "Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra" Journal of the ACM (JACM), Vol.42(1), pages:4366, 1995.

[13] P. van Beek, and R. Cohen. Exact and approximate reasoning about temporal relations. Computational intelligence, Vol 6(3), pp. 132147, 1990.

[14] B. Bennet. Knowledge Representation and Reasoning: Compositional Reasoning, 2007. Lecture notes, School of Computing, University of Leeds.

[15] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 web ontology language document overview. URL, 2009. http://www.w3.org/TR/owl2-overview/.

[16] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. SWRL: A semantic web rule language combining OWL and RuleML. URL, 2009. http://www.w3.org/Submission/SWRL/.

[17] Boris Motik, Ian Horrocks, and Ulrike Sattler. Representing Ontologies Using Description Logics, Description Graphs, and Rules. Artificial Intelligence, 7(2):7489, 2009.

[18] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 web ontology language Nwe Features and Rationale. URL, 2009. http://www.w3.org/TR/owl2-new-features/

[19] I. Horrocks, U. Sattler, A tableaux decision procedure for SHOIQ, in: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), Morgan Kaufman, 2005.

[20] F. Baader, U. Sattler, An overview of tableau algorithms for description logics, Studia Logica 69 (2001) 540.

[21] J. Renz "Maximal tractable fragments of the Region Connection Calculus: A complete analysis." Int. Joint Conference on Artificial Intelligence, vol. 16, pp. 448-455, 1999.

[22] P. van Beek, and R. Cohen "Exact and approximate reasoning about temporal relations." Computational intelligence, Vol 6(3), pp. 132147, 1990.

[23] D. Peuquet and C.-X. Zhan. 1987. An algorithm to determine the directional relationship between arbitrarily-shaped polygons in a plane. Pattern Recognition 20 : 65-74.

[24] Eric Prud'hommeaux, Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/

[25] C. Welty and R. Fikes: "A Reusable Ontology for Fluents in OWL". Frontiers in Artificial Intelligence and Applications, 150:226236, 2006.

[26] Natasha Noy, Alan Rector: Defining N-ary Relations on the Semantic Web. W3C Working Group Note, April 2006. http://www.w3.org/TR/swbp-n-aryRelations/

[27] http://jena.sourceforge.net/

[28] http://en.wikipedia.org/wiki/OpenMP

[29] http://www.adobe.com/products/photoshop.html