SOWL QL : Querying Spatio-Temporal Ontologies In OWL 2.0

Stravoskoufos Konstantinos

Master's thesis, April 25, 2013
Department of Electronic & Computer Engineering
Technical University of Crete (TUC)

Abstract

Querying spatio-temporal information in ontologies using query languages such as SPARQL leads to complicated queries and requires that the users be familiar with the underlying representation of spatio-temporal information. Adding spatial and temporal operators that hide the underlying representation from the end user while maintaining simplicity of expression in queries is an important issue to deal with.
We introduce SOWL QL, a high-level query language for querying spatio-temporal information in OWL ontologies. SOWL QL extends SPARQL with a powerful set of temporal and spatial operators, including Allen operators, and is capable of querying both quantitative and qualitative temporal and spatial information (e.g., information expressed using dates, times, temporal intervals or information expressed using natural language terms such as "before" or "after").
SOWL QL is part of SOWL, an approach for handling spatio-temporal information comprising of an ontology in OWL, the SOWL QL query language and a reasoner. Representation of dynamic concepts in SOWL is achieved using the "N-ary relations" or, alternatively, the "4D-fluents" mechanism. Both the 4D-fluents and N-ary mechanisms are thoroughly explored and expanded for representing qualitative (in addition to quantitative) spatio-temporal information in OWL. A spatial model representation has also been implemented in SOWL supporting the representation of both directional and topological relations. Moreover, SOWL offers reasoning support with a reasoner integrated within the ontology. Temporal and spatial reasoning in SOWL is realized by introducing a set of SWRL rules operating on spatial (topological or directional) relations as well as on temporal Allen relations.
As a proof of concept, SOWL QL is implemented in full along with a Graphical User Interface (GUI) facilitating ontology loading and parsing, query formulation and execution and, results viewing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of the World Wide Web (WWW) in recent years has generated the need for intelligent tools and mechanisms, which automatically handle tasks that are typically handled manually by users. For example, planning a trip requires booking and purchasing tickets at specific dates and prices, according to user needs. Buying a product requires careful selection among different products that satisfy user needs, at the best available price. All these tasks are handled by searching the Web (using a search engine). In recent years, there is an increasing need for Web services that accomplish these tasks automatically without user intervention, besides task description. These services must be capable of understanding the meaning of the content of Web pages and reason over their content in a way similar to humans. Semantic Web is a solution to this need by introducing formal, machine readable semantics for representing knowledge combined with reasoning and querying support. These form the basis of the Semantic Web initiative.

Formal definitions of concepts and of their properties form ontologies, which are defined using the OWL language. OWL ontologies offer the means for representing high level concepts, their properties and their interrelationships. For example medical ontologies represent the anatomic features of the human body, diseases, their symptoms and corresponding medical exams and treatments. Ontologies comprise of definitions of concepts and their properties by means of binary relations (i.e., between two concepts, or a concept and a numerical domain). Query languages such as SPARQL, are typically used for querying information in ontologies.

The syntactic restriction of OWL to binary relations complicates the representation of N-ary (e.g., ternary) relations. For example, an employment relation at a specific temporal interval that involves an employee, an employer and a temporal interval, is in fact a ternary relation. In general, properties of objects that change in time (dynamic properties) are not binary relations, since they involve a temporal interval in addition to the object and the subject. Representing and querying information evolving in time is the problem this work is dealing with.

1.1 Problem Definition

Dynamic ontologies are not only suitable for describing static scenes with static objects (e.g., objects in photographs) but also enable representation of events with objects and properties that change over time and space (e.g., moving objects in a video). Handling both static and dynamic information in the Semantic Web is an important problem to deal with. Representation of dynamic features calls for mechanisms that allow uniform representation of the notions of time (and of properties varying in time) within a single ontology. Existing methods for achieving this include, among others, temporal description logics [3], concrete domains [71], property labelling [43], versioning [58], named graphs [109], reification[1] and the 4D-fluents (perdurantist) approach [115].

Representing dynamic information in the Semantic Web is a complicated issue to deal with and is not handled in full by any of the aforementioned approaches. The syntactic restriction of OWL to binary relations complicates the representation of temporal properties since a property holding for a specific time instant or interval is a relation involving three objects (an object, a subject and a time instant or interval). Some methods (e.g., [3], [71]) require extending OWL with additional constructs and are not compliant with existing standards of the semantic Web for crafting, reasoning and querying ontologies.

Reasoning over qualitative spatial and temporal information (i.e., information defined using natural language expressions such as "before" or "left") is also a a requirement. For example, the description of a university campus using natural language involves expressions such as "north of", "into" instead of spatial coordinates.

In our previous work [17] we introduce SOWL, an approach for handling spatio-temporal information in OWL. The representation of spatio-temporal information in SOWL extends those of previous approaches for handling qualitative in addition to quantitative relations evolving in time and space while being compliant with OWL, existing tools and semantic Web standards. Apart from 4D-fluents, a representation of both forms of spatio-temporal information (i.e., quantitative, qualitative) based on N-ary relations [109] is also proposed. The two representations (i.e., 4D-fluents, N-ary relations) are practically equivalent with 4D-fluents being more suitable for the representation of symmetric, inverse and transitive relations using fewer additional relations, while the N-ary approach requires fewer additional objects. Reasoning in SOWL is implemented in SWRL and is capable of inferring spatial and temporal relations and detecting inconsistent assertions. Reasoning implements path consistency on tractable sets of spatial and temporal relations [10]. The mechanism is (besides soundness, completeness and tractability) compliant with existing W3C specifications, standards and tools. It is an integral

[1] http://www.w3.org/TR/swbp-n-aryRelations/

part of the ontology and is handled by standard tools such as Pellet.

The focus of the proposed work is on querying over spatio-temporal information in SOWL. Query support in SOWL is realized with the SOWL Query Language (SOWL QL), a high-level query language, independent from the underlying SOWL representation so that, the user need not be familiar with the peculiarities of the ontological spatio-temporal representation (i.e., the 4D-fluents or the N-ary approach). SOWL QL handles dynamic (spatio-temporal) ontologies almost like static ones and relies on the idea of extending SPARQL with spatio-temporal operators (i.e., SOWL QL builds-upon SPARQL, the current standard of the Semantic Web) that apply on the underlying SOWL representation.

1.2 Proposed Solution

The SOWL QL query language relies on the idea of extending SPARQL with spatio-temporal operators following the example of [109]. Compared to the work referred to above, SOWL QL has the following three advantages (a) supports both spatial and temporal operators, (b) is capable of querying over both quantitative and qualitative spatial and temporal information and (c) supports reasoning during the querying process. SOWL QL syntax is independent of the underlying ontological representation of spatio-temporal information. The working version of SOWL QL is implemented on top an N-ary relations representation, although a representation based on 4D-fluents has been implemented as well. Notice that, SOWL QL queries are translated into equivalent SPARQL queries, thus SOWL Query Language can be considered as a form of "syntactic sugar" over SPARQL for spatio-temporal queries.

SOWL QL uses the SOWL reasoner for answering queries specifying exact temporal or spatial values such as temporal instants, intervals or locations. This is a unique feature of SOWL QL not addressed by existing query languages. SOWL QL is capable of handling such cases by inserting any temporal values specified by the query into the knowledge base and by inferring all new relations (between the newly added values and the existing ones) using the SOWL reasoner, prior to answering the query.

1.3 Contributions of the Present Work

The contributions of the present work are summarized below:

- We introduce SOWL QL a language for querying over spatio-temporal information in OWL. Building upon SPARQL, SOWL QL is independent of the underlying spatio-temporal representation. An exhaustive set of spatial and temporal operators are defined in SOWL QL including timepoint, interval and

Allen temporal operators as well as topological and directional spatial operators. SOWL QL supports querying for both, quantitative and qualitative spatio-temporal expressions (i.e., information defined using natural language terms such as "before"). Queries specifying exact temporal or spatial values call for reasoning support, a feature that is also implemented within SOWL QL translation.

- For representing and reasoning over dynamic spatio and temporal information SOWL QL resorts to our previous work for SOWL [17].
- Graphical User Interface (GUI) operating on top of the SOWL QL interpreter for loading ontologies and executing queries SOWL QL is also implemented.
- The implementation of SOWL QL is available on the Web[2].

1.4 Thesis Outline

In Chapter 2 we discuss related work in the field of knowledge representation and Query languages. Earlier work related to representation, reasoning and querying over temporal and spatial information is presented and discussed. In Chapter 3 we present the proposed SOWL ontology model for spatio-temporal information. In this chapter we discuss two temporal models, the 4D-fluents and the N-ary relations model, along with a model for representing spatial information. The combination of spatial and temporal representations is also discussed here. In Chapter 4 we present the corresponding reasoning mechanism over point-based and interval based temporal representations as well as the spatial reasoning. Chapter 5 presents the SOWL Query Language. In this chapter we discuss language syntax and semantics, all spatial and temporal operators and examples of SOWL QL queries. Next, in Chapter 6, we present the implementation looking in more detail at the different parts of the system and finally, in Chapter 7 we discuss conclusions and issues for future work.

[2]http://www.intelligence.tuc.gr

Chapter 2

Background and Related Work

Semantic Web standards such as ontologies, ontology construction languages, reasoning and rules are discussed in Section 2.1. Related work on the field of temporal and spatial representation is presented in Section 2.2 followed by related work on reasoning in Section 2.3. Existing work representing temporal and spatial information in ontologies as well as on reasoning on temporal and spatial representation, are discussed in Section 2.4 and 2.5 respectively. Besides representation and reasoning, querying support for spatio-temporal informations is provided by means of spatio-temporal query languages. Related work general Semantic Web query languages as well as, query languages for spatio-temporal information are presented in Section 2.6.

2.1 Semantic Web

The rapid growth of available information in the World Wide Web (WWW) has complicated the task of information retrieval and processing over the Web. Search engines such as Google[1], Bing[2] and Ask[3] facilitate retrieval tasks, however in most cases, users still have to browse through the returned Web pages in order to fulfil tasks such as on-line shopping, travel planning or ticket reservations. Advanced search mechanisms may also be implemented based-upon machine learning or focused crawlers [25, 16] but still are incapable of fully automating tasks such as those referred to above. Automating these tasks require that machine understandable semantics become available and usable along with data existing already in HTML pages that are readable by humans as well as by machines. The requirement of machine interpreted Semantics is the core idea of the Semantic Web vision [19].

[1] http://www.google.com
[2] http://www.bing.com
[3] http://www.ask.com

Introducing machine readable semantics calls for a formal language for conceptualization of application domains, the related concepts, their properties and their relationships. Based-upon existing work on knowledge representation, logic and ontologies along with more recent approaches such as frames [78], Description Logics [4] form the basis for Semantic Web standards. Specifically OWL [77] and SWRL [50] are the description and rule languages respectively of the Semantic Web and are presented in following.

2.1.1 Description Logics

Description Logics (DLs) [5] is a fragment of First Order Logic (FOL) composing the basis for the Semantic Web standards. They are related to existing formalisms such as propositional and first order logic [34] and frames [78]. Compared to frames, it adobpts formal semantics and is more expressive than propositional logic but, less expressive (in order to increase performance) compared to First Order Login (FOL).

The basic components of a Description Logic formalism are the *concepts* or *classes*, their *properties* or *roles* and the *individuals* or *objects*. The expressiveness of a description logic formalism is defined by the allowable constructs and expressions. The trade-off between expressive power and efficiency is the basic design decision in every DL. Every DL supports a different set of allowable expressions for defining concepts. These definitions are expressed as combinations of existing definitions and atomic (simple) concepts and of their properties. The formal, logic-based semantics of Description Logics allow for unary predicates (concepts) and binary predicates (roles or properties).

Besides representation of concepts and of their properties, DL formal and unambiguous semantics allow for inference of implied facts from asserted knowledge. The expressive power of DLs is complemented by inference procedures dealing with *subsumption* (i.e., determining subclass-superclass relations), *consistency* (i.e., determining contradictions in concept definitions and individual assertions) and *instance* (i.e., determining the class(es) that an individual belongs to). Decidability of inference is a highly desirable characteristic so that, in practice, expressiveness is often sacrificed (i.e., restricted) in order to guarantee decidability. The OWL language is based on DL and it is the basic component of the Semantic Web initiative.

A description logic, or language, is fully characterized by the allowable constructs that are used for the definitions of concepts and properties, as expressions of basic (atomic) concepts and properties. The set of such definitions for an application domain forms the Terminological Box (TBox) of an ontology. Assertions involving concepts and properties of individuals form the Assertional Box (ABox) of the ontology. Reasoning is applied on both, TBox definitions and ABox assertions.

The basic description language is the $\mathcal{ALC}$ language which allows for concept negation, intersection and union. Specifically, concepts forming a set, abbreviated

as N_c, and properties (roles) forming the set N_r can be defined. Atomic concepts and properties are part of N_c and N_r respectively. The negation of a concept C (atomic concept or description) abbreviated as $\neg C$, the intersection and union of concepts C and D (atomic concepts or descriptions), abbreviated as $C \sqcap D$ and $C \sqcup D$ respectively are also defined. Finally, if C is a concept then $\exists r.C$ (i.e., objects which relate with property r with at least one individual belonging to class C) and $\forall r.C$ (i.e., class of individuals that property r relates them only with individuals of class C), are also allowable $\mathcal{ALC}$ concept definitions. They are called *existential* and *value restrictions* respectively.

Formally given a set Δ^I, an *interpretation* I $(\Delta^I, .^I)$ is a pair Δ^I and function $.^I$ that assigns to every concept A^I the set $A^I \subseteq \Delta^I$ and to every property r the set $r^I \subseteq \Delta^I \times \Delta^I$. The extension of interpretation I for $\mathcal{ALC}$ complex concepts is defined as follows:

- $(C \sqcap D)^I = C^I \cap D^I$
- $(C \sqcup D)^I = C^I \cup D^I$
- $(\neg C)^I = \Delta^I \setminus C^I$
- $(\exists r.C)^I = \{x \in \Delta^I |$ exists $y \in \Delta^I$ and $(x, y) \in r^I$ and $y \in C^I\}$
- $(\forall r.C)^I = \{x \in \Delta^I |\ \forall y \in \Delta^I, (x, y) \in r^I \Rightarrow y \in C^I\}$

Besides concept *conjunction, disjunction* and *negation*, $\mathcal{ALC}$ description language includes the *top concept* $\top$ which is an abbreviation of the whole domain and $\bot$ *bottom concept* which represents the empty set. ABox assertions can be the following:

- $C(a)$ where a is an individual and C a concept
- $r(a, b)$ where a, b are individuals and r a property

An important part of a Description Logic expressiveness are *General Inclusion Axioms* that specify that a concept C is subsumed by another concept D (i.e., all individuals of C also belong to D. Formally:

- $C \sqsubseteq D \iff C^I \subseteq D^I$

Equivalence axioms $C \equiv D$ correspond to axioms: $C \sqsubseteq D$ and $D \subseteq C$. Additional constructs not in $\mathcal{ALC}$ but parts of more expressive description logics are *qualified number restrictions*: they can be either $(\geq nr.C)$ (*at least* restriction) or $(\leq nr.C)$ (*at most* restriction) implying that individuals of a concept can be related with property r with at least (or at most) n individuals of concept C with property r:

- $(\geq nr.C)^I = \{d \in \Delta^I | \ cardinality(\{e|(d,e) \in r^I \wedge e \in C^I\}) \geq n\}$
- $(\leq nr.C)^I = \{d \in \Delta^I | \ cardinality(\{e|(d,e) \in r^I \wedge e \in C^I\}) \leq n\}$

The *unqualified number restrictions* are defined as the qualified ones but concept C in the corresponding definition is replaced by the top concept $\top$. The *exact cardinality restrictions* are defined as the result of a combination of an *at least n* and an *at most n* restriction. Exact value restrictions force a specific value for a property of individuals of a concept and *nominal* concepts whose individuals can be one of a restricted list of specific individuals. *Inverse properties* r, r^- are also defined so that when r holds between two individuals r^- also holds but with the object and subject of the property interchanged:

- $(r^-)^I = \{(b,a)|(a,r)\epsilon r^I\}$

If a property r is the inverse of itself then r is *symmetric*. A $functional$ property represents *at most one* restriction over the property and $inversefunctional$ a property that its inverse is functional. A property r is *transitive* when:

- $(r(a,b) \wedge r(b,c) \Rightarrow r(a,c)$

General role inclusion axioms are an important part of the expressiveness of a description logic, given a set of properties (roles) $r_1, r_2, ..., r_n, r$ then:

- $r_1 \circ r_2 ... \circ r_n \sqsubseteq r \Leftrightarrow r_1(a_1,a_2) \wedge r_2(a_2,a_3) ... r_n(a_n,a_{n+1}) \Rightarrow r(a_1,a_{n+1})$

where $\circ$ denotes composition. Limited forms of general role inclusion axioms are *subproperty axioms* (i.e., property r_1 is subproperty of property r_2 when $r_1(a,b) \Rightarrow r_2(a,b)$). Also *reflexive* properties (i.e., r is reflexive when $r(a,b) \Rightarrow r(a,a)$), $irreflexive$ properties (i.e., r is irreflexive when $\forall a : r(a,a) \sqsubseteq \neg r^I$) and $disjoint$ properties (i.e., properties r_1, r_2 are disjoint when $r_1^I \sqsubset \neg r_2^I$ and $r_2^I \sqsubset \neg r_i^I$) are also specific forms of role inclusion axioms. A description logic may also offer support of specific *datatypes* such as integer numbers and strings. The notation characterizing the expressiveness of a description logic can be summarized as follows:

- $\mathcal{F}$: Functional properties
- ε:qualified existential restrictions
- $\mathcal{U}$:Concept union
- $\mathcal{C}$:Negation (including non atomic concepts)
- $\mathcal{S}$: $\mathcal{ALC}$ with transitive properties
- $\mathcal{H}$: Subproperty axioms

- $\mathcal{R}$: reflexive, irreflexive and disjoint properties
- $\mathcal{O}$: nominals
- $\mathcal{I}$:inverse properties
- $\mathcal{N}$: Unqualified cardinality restrictions
- $\mathcal{Q}$: Qualified cardinality restrictions
- ($\mathcal{D}$):datatypes
- $\mathcal{AL}$: concept negation, intersection, value restrictions and limited (unqualified) existential restrictions.
- $\mathcal{EL}$:intersection and full existential restrictions
- $\mathcal{FL}_o$:$\mathcal{FL}^-$ without existential qualification

The expressiveness of a description language complicates reasoning tasks, so that description logic expressiveness and efficiency of reasoning tasks are always traded-off. Also, the following reductions apply to reasoning tasks, and have polynomial time complexity [5]:

- Subsumption is reduced to equivalence
- Equivalence can be reduced to subsumption
- Subsumption can be reduced to satisfiability
- Satisfiability can be reduced to subsumption
- Satisfiability can be reduced to consistency
- Instance problem can be reduced to consistency
- Consistency can be reduced to the instance problem

For terminological (TBox) reasoning, implementing an algorithm for satisfiability is sufficient for all reasoning tasks (i.e., satisfiability, equivalence and subsumption). Accordingly, implementing a consistency algorithm is sufficient for implementing assertional (ABox) reasoning (i.e., consistency and instance). Description logics are a fragment of First Order Logic and resolution based approaches (i.e., a reasoning method for first order logic) where initially employed for the required reasoning tasks. Recently, a shift towards the so called “tableaux” based reasoning

is observed [5]. Popular reasoners such as FaCT++[4], Pellet[5], Hermit[6] and RACER[7] are examples of tableaux based reasoners.

Description logics do not adopt the *Unique Name Assumption* and the *Closed World Assumption* as the Entity Relationship Model [28] does. Two individuals with different names are not considered distinct unless this is asserted or proved by the asserted axioms. DLs adopt the so called *Open World Assumption* implying that asserted knowledge is not considered to be complete, thus failing to prove a fact does not mean that the fact does not hold true. Only proven facts can be asserted into the knowledge base and lack of a proof does not imply proof of the negation of the fact.

2.1.2 OWL

The objective of Semantic Web standards is to offer the means for formal machine understandable semantics of application domains. This formal conceptualization, or ontologies is based on the OWL language introduced initially in [51]. Building-upon DAML [2] and OIL [35], OWL was compatible with the existing RDF [69] specification for describing concepts and properties of objects, while offering increased expressiveness over the RDFS [76] vocabulary description language for RDF.

RDF and RDFS represent properties or relations between entities in the form of triplets of the form *object-predicate-subject* (e.g., IBM employs John). Specific individuals can belong to classes (e.g., John is-a Person, where *John* is an individual and *Person* is a class). Properties such as *employs* can relate individuals of specific classes, for example one can specify that the object of the property *employs* belongs to class *Company* and the subject belongs to the class *Employee*. Classes of the object and the subject of a property are abbreviated as *domain* ans *range* respectively. Basic taxonomic relations between classes and properties can be also specified, for example is can be stated that *Employee* is a *subclass* of *Person* (i.e., every employee is also a person). OWL extends RDF/RDFS expressiveness and it will be described in detail in the following. Three variants of OWL where introduced in the OWL specification [18]:

- OWL-Full which is fully compliant with RDF,
- OWL-DL which is based on Description Logics and,
- OWL-Lite is a subset of OWL-DL, it is less expressive than OWL-DL allowing for the definition of class hierarchies and simple constraint features.

[4] http://owl.man.ac.uk/factplusplus/
[5] http://clarkparsia.com/pellet/
[6] http://www.hermit-reasoner.com/
[7] http://www.sts.tu-harburg.de/ r.f.moeller/racer/

OWL-Full is the most expressive variant but does not retain decidability and it is not supported by existing OWL reasoners. OWL-DL is the most routinely used version of OWL and it is based in the $\mathcal{SHOIN}(\mathcal{D})$ description logic [51]. Thus, OWL DLs available constructs are those of $\mathcal{SHOIN}(\mathcal{D})$: Concept negation, union, intersection, value and existential restrictions and transitive properties ($\mathcal{S}$), subproperties ($\mathcal{H}$), nominals ($\mathcal{O}$), inverse properties ($\mathcal{I}$), unqualified cardinality restrictions ($\mathcal{N}$) and datatypes. Reasoning over $\mathcal{SHOIN}(\mathcal{D})$ is non deterministic exponential in time (NExpTIME) although, in practice, optimised tableaux based reasoners [5] offer tractable average case running times. OWL-Lite is based on $\mathcal{SHIF}(\mathcal{D})$ description logic [77]. OWL-Lite ($\mathcal{SHIF}(\mathcal{D})$) supports concept negation, union, intersection, value and existential restrictions and transitive properties ($\mathcal{S}$), subproperties ($\mathcal{H}$), and inverse properties ($\mathcal{I}$), it doesn't support nominals and unqualified cardinality restrictions as OWL-DL does, but it supports functional properties ($\mathcal{F}$), a limited form of cardinality restrictions. Reasoning over OWL-Lite is deterministic exponential in time although, average case complexity, as in the case of OWL-DL, is much lower in practice [5].

The evolution of the OWL specification was based on the observation that additional constructs can be added in OWL-DL without compromising decidability, while increasing expressiveness. Extending OWL-DL with the additional constructs leaded to the adoption of OWL 2 [39] as the current Semantic Web standard [82]. OWL 2 is based on $\mathcal{SROIQ}(\mathcal{D})$ description logic [49] (i.e., $\mathcal{SROIQ}(\mathcal{D})$ offers all constructs of $\mathcal{SHOIN}(\mathcal{D})$ with the addition of qualified number restrictions ($\mathcal{Q}$) and complex role inclusion axioms ($\mathcal{R}$)). The computational properties of $\mathcal{SHROIQ}(\mathcal{D})$-*OWL 2.0* are analysed in [49], along with a description of the corresponding tableaux algorithm for reasoning over OWL 2. In addition to constructs offered by OWL-DL, OWL 2 offers support for qualified number restrictions ($\mathcal{Q}$) in addition to the unqualified ones and, complex role inclusion axioms ($\mathcal{R}$) along with disjoint, symmetric, asymmetric, reflexive, irreflexive properties and property negation in addition to subproperties offered by OWL-DL. $\mathcal{SHROIQ}(\mathcal{D})$ is decidable thus offering additional expressiveness, while retaining the computational properties of OWL-DL.

OWL 2 specification includes *profiles*[8], namely *OWL 2* $\mathcal{EL}$, $\mathcal{QL}$ and $\mathcal{RL}$. *OWL 2* $\mathcal{EL}$ supports class conjunction and existential restrictions while disallowing negation, disjunction, cardinality and value restrictions, in order to optimize classification tasks. *OWL 2* $\mathcal{EL}$ offers tractable (i.e., polynomial time) reasoning performance. *OWL 2* $\mathcal{QL}$ is offering optimized performance for conjunctive query answering by allowing class disjointness, domain, range of properties, existential restrictions, disallows disjunctions and value restrictions. *OWL 2* $\mathcal{RL}$ is optimized for rule-based reasoning over individuals explicitly asserted into the ontology, thus

[8] http://www.w3.org/TR/owl2-profiles/

disallowing constructs such as existential restrictions that introduce anonymous individuals.

OWL specification also includes the *abstract syntax* which is equivalent to the *Description Logic based syntax* presented in section 2.1.1. Specifically, class names (usually with a capital initial letter) represent concepts and keywords. For example, $owl:Thing$ and $owl:Nothing$ represent the top $\top$ and bottom $\bot$ concepts respectively. $intersectionOf(C_1, C_2...C_n)$ and $unionOf(C_1, C_2...C_n)$ represent the intersection and disjunction of classes $C_1, C_2...C_n$. The complement of a class C is defined using keyword $complementOf(C)$ while an enumerated class defined as a set of individuals $o_1, o_2...o_n$ is defined using the declaration $oneOf(o_1...o_n)$.

Restrictions (expressed using the *restriction* keyword followed by the property P over which the restriction applies and the restriction keyword) can be one of the following: *someValuesFrom(C), allValuesFrom(C), hasValue(o), minCardinality(n)* and *maxCardinality(n)* representing qualified existential restrictions, value restrictions, exact value restriction, min and max cardinality restrictions respectively, where C is a class name, o an individual (or datatype value), and n an integer. An enumeration using the $oneOf$ keyword can be used instead of a class name C in the above definitions. TBox class definitions can be *A partial* $C_1..C_n$ (i.e., class A is a subclass of the conjunction of $C_1...C_n$), and *A complete(*$C_1...C_n$*)* (i.e., A equals the intersection of $C_1...C_n$). Enumerated classes are defined using the *EnumeratedClass(A* $o_1...o_n$*)* keyword (where $o_1...o_n$ are individuals). *SubClassOf(*C_1 C_2*)* asserts that C_1 is a subclass of C_2, while the $EquivalentClasses(C_1...C_2)$ and $DisjointClasses(C_1...C_n)$ define class equivalence and disjointness for the list of class names.

Keyword $SubPropertyOf(p_1\ p_2)$ defines the subproperty relation between properties p_1 and p_2, while property equivalence is defined using *EquivalentProperties* keyword. Domains and ranges are defined using *domain* and *range* keywords with class definitions as arguments, while keywords *inverseOf, Symmetric, Asymmetric, Functional, InverseFunctional, Transitive, DisjointProperties, Reflexive* and *Irreflexive* apply on properties having the obvious interpretations. Finally, *SameIndividual* and *DifferentIndividuals* apply on lists of individuals specifying their equivalence or difference respectively. Both, abstract syntax and Description Logic based syntax will be used in this thesis.

2.1.3 SWRL

SWRL[9] is the language for specifying rules applying on Semantic Web ontologies. *Horn Clauses* (i.e., a disjunction of classes with at most one positive literal), can be expressed using SWRL, since Horn clauses can be written as implications (i.e.,

[9] http://www.w3.org/Submission/SWRL/

$\neg A \vee \neg B ... \vee C$ can be written as $A \wedge B \wedge ... \Rightarrow C$). The efficiency of reasoning over Horn clauses using forward chaining algorithms is a reason for choosing this form of rules. The antecedent (body) of the rule is a conjunction of clauses. Notice that, neither disjunction nor negation of clauses is supported in the body of rules. Also, the consequence (head) of a rule is one positive clause. Again, neither negation nor disjunction of clauses can appear as a consequence of a rule. Conjunction of clauses into the consequence part of the rule can be expressed indirectly by a set of rules with identical antecedents.

The clauses in the rule can be class names (e.g., C) with variables as arguments (e.g., $C(?x)$), property names p (in the form $p(?x, ?y)$ where x, y are variables) or the OWL $sameAs$, $differentThan$ keywords. Specific build ins can be also supported for numerical operators supporting specific datatypes. Whenever the antecedent of the rule holds for a given set of variable instantiations, the consequence is asserted into the knowledge base. To guarantee decidability, the rules are restricted to be *DL-safe rules* [81] that apply only on named individuals in the ontology ABox and not on implied anonymous individuals. Even with the syntactic restrictions imposed on SWRL rules, such as the lack of negation and disjunction and their applicability only to ABox reasoning, SWRL still extends OWL expressiveness while retaining decidability. For example, intersection of properties over named individuals can be expressed using SWRL although, this is not part of OWL specification. Therefore, SWRL is an important tool for embedding rules into an ontology, while retaining decidability and OWL semantics. In this thesis, SWRL rules are presented using a first order notation in place of its equivalent SWRL notation.

2.1.4 SPARQL

SPARQL [88] is the W3C recommendation query language for RDF. The basic evaluation mechanism for SPARQL queries is based on graph matching. The query criteria are given in the form of RDF triples possibly with variables in the place of the subject, object, or predicate of a triple, called basic graph patterns. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middle ware. SPARQL is capable of querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, nested queries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs. The SPARQL query language structure is illustrated in figure 2.1.

Prologue (Optional)	BASE ‹uri› PREFIX prefix:‹uri› (Repeatable)
Query Results Form (Required)	SELECT (DISTINCT) ?Var1 ?Var2 .... ?VarN SELECT (DISTINCT) * DESCRIBE ?Var1 ?Var2 .... ?VarN or ‹uri› DESCRIBE * CONSTRUCT { Graph Pattern } ASK
Query Dataset Sources (Optional)	FROM ‹uri› FROM NAMED ‹uri›
Graph Pattern (Optional, Required for ASK)	WHERE { Graph pattern [Filter Expression] }
Query Results Ordering (Optional)	ORDER BY
Query Results Selection (Optional)	LIMIT n , OFFSET m

Figure 2.1: SPARQL query language structure

Looking in more detail at the structure of SPARQL:

- PREFIX: Here we declare short abbreviations for the namespaces used in a query.
- SELECT: The SELECT clause defines a list of variables bound in a query pattern. The syntax *SELECT ** is an abbreviation that selects all of the variables in a query.
- FROM: Defines the data set to be queried. The FROM clause is optional.
- CONSTRUCT: It is used to return an RDF graph created by substituting variables in the query pattern.
- DESCRIBE: It is used to return an RDF graph describing the resources that were found.
- ASK: It returns a boolean value indicating whether the query pattern is matched or not.

- WHERE: This clause provides the basic graph pattern to match against the data graph. Basic graph patterns are sets of triple patterns. A triple is a data entity composed of subject-predicate-object, like “Bob is 35” or “Company hasEmployee George”. The WHILE clause may include optional triples. If the triple to be matched is optional, it is evaluated when it is present, but the matching does not fail when it is not present. Also, it is possible to make UNION of multiple matching graphs - if any of the graphs matches, the match will be returned as a result.

- FILTER: In addition to specifying graphs to be matched in the WHILE clause, constraints can be added for values using the FILTER construct. By using this construct we can apply all kinds of value restrictions such as string value restrictions (like FILTER regex(?name, “company”) meaning that the ?name variable must contain the string “company” as a substring) or number value restrictions (like FILTER (?salary <300) meaning the ?salary must be less than 300) Also, a few special operators are defined for the FILTER construct. These include the “isIRI” operator for testing whether a variable is an IRI/URI, the “isLiteral” operator for testing whether a variable is a literal and “bound” to test whether a variable is bound to other variables.

- DISTINCT: Used to distinguish the unique results.

- ORDER BY: The ORDER BY clause establishes the order of a solution sequence. Following the ORDER BY clause is a sequence of order comparators, composed of an expression and an optional order modifier (either ASC() or DESC()).Each ordering comparator is either ascending (indicated by the ASC() modifier or by no modifier) or descending (indicated by the DESC() modifier).

- LIMIT: The LIMIT clause puts an upper bound on the number of solutions returned. If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned.

- OFFSET: OFFSET causes the solutions generated to start after the specified number of solutions. An OFFSET of zero has no effect. Using LIMIT and OFFSET to select different subsets of the query solutions will not be useful unless the order is made predictable by using ORDER BY.

An example of a SPARQL query is illustrated in Figure 2.2.

SPARQL EXAMPLE QUERY

Data :

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query :

```
SELECT  ?title
WHERE
{
  <http://example.org/book/book1>  <http://purl.org/dc/elements/1.1/title>  ?title .
}
```

Result :

title
"SPARQL Tutorial"

Figure 2.2: SPARQL example

For spatio-temporal querying, SPARQL requires the user to be familiar with the underlying ontology model implemented (e.g., the 4D-fluents or the N-ary relations model in this work). Moreover, matching the spatio-temporal representation graph using RDF triples may require very complicated queries. Figure 2.3 is an example of a temporal query in SPARQL using the 4D-fluents model for the representation of temporal informaiton in an ontology. This query would return all the employees that work for a specific company at any time. Querying for temporal information requires using constructs specific to the 4D-fluents model (e.g., tsTimeSliceOf, tsTimeInterval) so that the user need to be familiar with this model for expressing temporal queries. This is an inherent limitation to SPARQL (which is not originally defined for expressing temporal queries, the same as spatial queries as we shall see in the following) and this is exactly the problem this work is dealing with.

```
SELECT ?employee
WHERE {
?timeSlice_0 ex:hasEmployee ex:Company1.
?timeSlice_0 4dfluents:tsTimeInterval ?interval_0.
?timeSlice_0 ex:hasEmployee ?timeSlice_1.
?timeSlice_1 4dfluents:tsTimeSliceOf ?employee.
?timeSlice_1 4dfluents:tsTimeInterval ?interval_0
}
```

Figure 2.3: SPARQL temporal query using the 4D-fluents model for temporal representation

2.2 Representation of Time and Space

Space and time are fundamentals aspects of the conceptualization of the physical world. The notions of time and space as well as the evolution of concepts and individuals in space and time are important issues in almost all application domains.

Time can be regarded as discrete or continuous, linear or cyclical, absolute or relative, qualitative or quantitative. Also, time can be presented using time instances or intervals. Temporal concepts as used by humans in every day life are represented in the Semantic Web using (in many cases) the OWL-Time ontology [48]. OWL-Time provides definitions of time instants (or points), intervals, definitions for their relations and definitions of concepts such as days, weeks, months, years, dates, time zones, durations and measuring units. OWL-Time is an ontology of the concepts of time, but OWL-Time does not specify how these concepts can be used to represent evolving properties of objects (i.e., properties that change in time) and it does not specify how to reason over qualitative relations of temporal intervals and instants. This is a problem this work is dealing with. Nevertheless, since OWL-Time is a W3C recommendation (although other ontologies such as the SWRL Temporal Ontology[10] also exist), it is adopted by the current work for providing definitions of the concepts of time. In addition, this work will show how these concepts can be related to dynamic concepts evolving in time.

There is a fundamental philosophical controversy concerning the evolution of concepts in time, namely the perdurantist and the endurantist approaches [101] and this controversy also applies to the generalized spatio-temporal representation [41]. A discussion on these issues from the philosophical standpoint can be found in [47, 113]. This controversy is related to issues such as the identity of objects as they evolve in time and whether objects endure in time although their properties may change, implying a fundamental distinction between objects and events or they

[10]http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl

perdure in time by relating properties in time and space in a form of a generalized event (4D perdurantist approach).

According to the perdurantist approach, every object is in fact a generalized event having specific spatial and temporal extensions. Even objects such as the sun and the solar system can be regarded as temporal entities having a specific duration, not different in their fundamental aspects from everyday events. In this work, both approaches are taken into account and modelling approaches based on 4D objects and 3D objects participating into events are presented. In this work, time is described using quantitative or qualitative terms using temporal instances and intervals. The employed representations are based on absolute time specifying an ordering of time points.

Choosing between a point or an interval-based representation is an important issue [110]. Point-based representations assume linear ordering of time points with three possible relations the "$<$", "$>$", "$=$" often referred to as *before, after* and *equals* respectively. Based on these ordering relations, intervals can also be defined as ordered pairs of points s, e with $s < e$, often referred to as *start* and *end* of an interval respectively. Given two such ordered pairs of points (i.e., intervals) the following relations can be defined between the two intervals i_1, i_2 in terms of their endpoints s_1, e_1 and s_2, e_2 respectively:

$$
\begin{aligned}
&i_1\ before\ i_2 \equiv e_1 < s_2\\
&i_1\ equals\ i_2 \equiv s_1 = s_2 \wedge e_1 = e_2\\
&i_1\ overlaps\ i_2 \equiv s_1 < s_2 \wedge e_1 < e_2 \wedge e_1 > s_2\\
&i_1\ meets\ i_2 \equiv e_1 = s_2\\
&i_1\ during\ i_2 \equiv s_1 > s_2 \wedge e_1 < e_2\\
&i_1\ starts\ i_2 \equiv s_1 = s_2 \wedge e_1 < e_2\\
&i_1\ finishes\ i_2 \equiv s_1 > s_2 \wedge e_1 = e_2
\end{aligned}
$$

The relations *after, overlappedby, metby, contains, startedby and finishedby* are the inverse of *before, overlaps, meets, during, starts and finishes* and are defined accordingly (by interchanging s_1, s_2 and e_1, e_2 in their respective definitions). A temporal relation can be one of the 13 pairwise disjoint Allen's relations [1] of Figure 2.4.

Using either a point or an interval-based representation, qualitative relations can be asserted, even when the specific time points or the temporal extends of intervals are unknown but their relative position is known; thus, the expressiveness of the representation increases. Quantitative representations involve specific datatypes such as $xsd : date$ supported by OWL, and these datatypes support comparison of dates, thus yielding the required ordering relation when specific dates of points are known.

Space is an important aspect of knowledge representation. Space can be regarded as two dimensional (2D) or three dimensional (3D) with most applications adopting

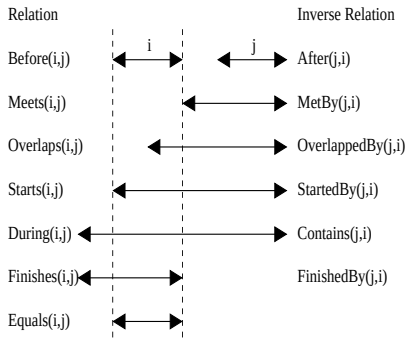

Figure 2.4: Allen's Temporal Relations

a 2D space, which is the approach followed in this thesis. Cartesian coordinates either 2D (x, y) or 3D (with z being the third axis) are employed.

Our approach is suitable for most applications, although there are limitations when a 2D projection is not adequate, for example in applications that handle earth in a global scale and not as a set of planes representing local areas. Each point is represented using a pair (or a triple) of coordinates of a specific datatype related to the coordinate system and the scale employed. Regions are represented using a set of points representing their contour or using minimum bounding rectangles (i.e., the minimal rectangle with sides parallel to the axis that contain the space of the region in question). Spatial information can be *topological, directional* and *distance* [93].

Topological relations represent the relative position of regions in the plane. The most widespread formalism for representing such relations is the so called Region Connection Calculus (RCC) Formalism introduced in [90]. A form of this calculus specifying 8 possible mutually exclusive relations between two regions, called RCC8 calculus is the most commonly used. The topological relations shown in Figure 2.5, *(DC, EC, EQ, NTPP, NTPPi, TTP, TPPi, PO)*, referred to as RCC8 relations are supported in the SOWL model.

Directional relations are also defined based on cone-shaped areas [79]. As shown in Figure 2.6, eight directional relations can be identified namely North (N), North East (NE), East (E), South East (SE), South (S), South West (SW), West (W) and North West (NW) following the cone-shaped regions approach of [79].

Alternative approaches based on 2D projections are discussed in [93, 103] and are also supported as well. The projection-based approach handles the projections of points (or regions) over the coordinate system axis, as shown in Figure 2.7. In case of points, these projections are single point projections on two axes while, in case of regions these projections form a two-point projection (or interval) on each axis. These projections are in turn handled the same way as point and interval

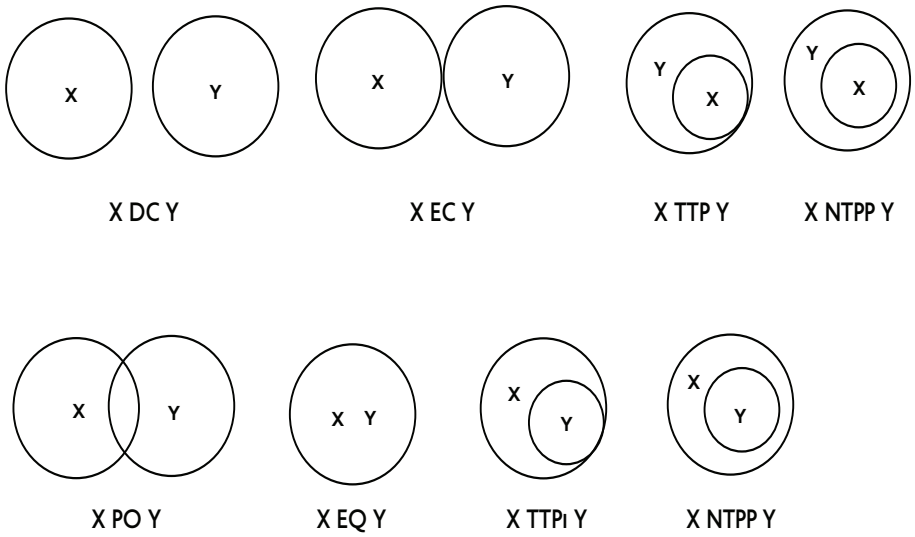

Figure 2.5: RCC8 topologic relations.

algebra handles time instants in the case of a temporal representation.

Finally, distance relations are defined and can be used in SOWL in conjunction with the above relation types. Notice that, qualitative distance relations (e.g., "far" and "near") may be ambiguous especially in applications where a common scale for measuring distances is not provided [93]. This problem is resolved when distance relations are expressed quantitatively (e.g., 3Km away from city A) and stored in the ontology as N-ary relations (i.e., by defining an object with attributes the two related locations and a numerical attribute representing their distance). In SOWL, we opt for the later (quantitative) approach for representing distance information.

2.3 Temporal and Spatial Reasoning

Inferring implied relations and detecting inconsistencies are handled by a reasoning mechanism. In the case of a quantitative representation such a mechanism is not required because spatial and temporal relations are extracted from the numerical representations in polynomial time (e.g., using datatype comparisons in the case of temporal relations and computational geometry algorithms in the case of spatial relations).

In cases where relations are qualitative, assertions of relations holding between spatial and temporal entities (e.g., intervals, points) restrict the possible assertions holding between other temporal and spatial entities in the knowledge base. For example, the assertion "point A is *north of* point B" impose a restriction on the arrangement of points is space. Also, it imposes a restriction on future assertions

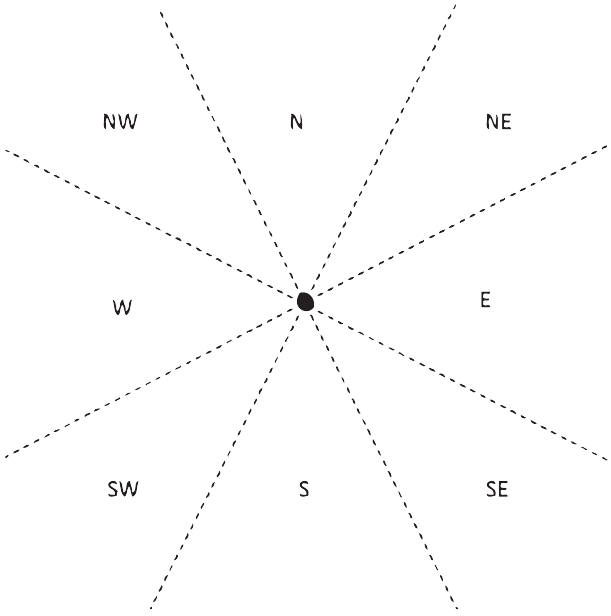

Figure 2.6: Cone-based direction relations.

(e.g., a new assertion such as: "A is *south of* B" contradicts the existing one). Then, reasoning on qualitative spatial or temporal qualitative relations can be transformed into a *constraint satisfaction problem*, which is known to be an *NP-hard* problem in the general case [93]. The worst case complexity appears in certain instances that are neither over-constrained nor under-constrained [93]. Notice that, a large number of constraints (i.e., assertion of relations) in comparison to the number of entities involved, usually leads to an inconsistency and a small number of constraints usually imply a small number of implied relations; both cases are are resolved in polynomial time complexity in practice [93].

Reasoning over qualitative spatial and temporal relations is achieved using [93]:

- An exponential worst case algorithm that has better performance on the average case.
- Approximation algorithms that are neither complete nor sound but they have polynomial worst case complexity.
- Polynomial time algorithms that are sound and complete by restricting the allowable relations to specific tractable sets.

Inferring implied relations depends on existing relations in the knowledge base and on their semantics. For example, directional relations may have different semantics if they are interpreted using the cone-shaped approach [79] or the projection-based approach [7, 103]. Although, the relations in both cases are the same, their

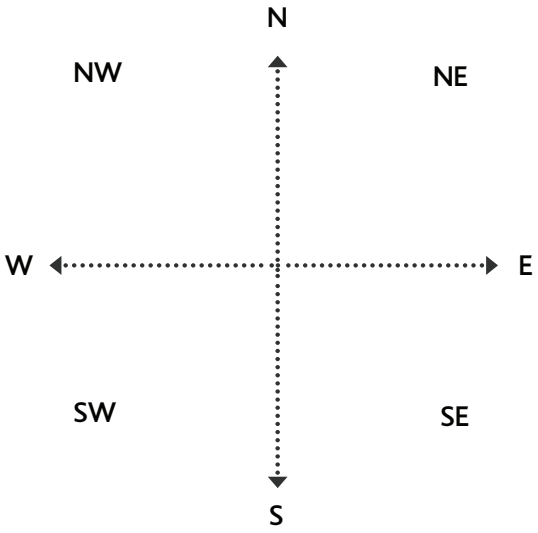

Figure 2.7: Projection-based direction relations.

interpretations and semantics differ. Inferring implied relations is achieved by specifying the result of *compositions* of existing relations. Specifically, when a relation (or a set of possible relations) R_1 holds between entities A and B and a relation (or a set of relations) R_2 holds between entities B and C then, the *composition* of relations R_1, R_2 (denoted as $R_1 \circ R_2$) is the set (which may contain only one relation) R_3 of relations holding between A and C. Typically, compositions of pairs of relations are stored in *composition tables* [93].

Qualitative relations under the intended semantics may not apply simultaneously between a pair of individuals. For example, given time instants p_1 and p_2, p_1 can not be simultaneously $before$ and $after$ p_2. Typically, in spatio-temporal representations (e.g., using Allen and RCC8 relations) all basic relations (i.e., simple relations and not disjunctions of relations) are pairwise disjoint. When disjunctions of basic relations hold simultaneously then, their set intersection holds. For example, if p_1 is *before or equals* p_2 and simultaneously p_1 is *after or equals* p_2 then p_1 *equals* p_2. In case the intersection of two relations is empty these relations are disjoint. Checking for consistency means checking if asserted and implied relations are disjoint.

Reasoning over spatio-temporal relations is known to be an NP-hard problem and identifying tractable cases of this problem has been in the center of many research efforts over the last few years [93]. The notion of *k-consistency* is very important in this research. Given a set of n entities with relations asserted between them imposing certain restrictions, *k-consistency* means that every subset of the n entities containing at most k entities does not contain an inconsistency. Obviously, when *n-consistency* holds then, there is not an inconsistency, but checking for all

subsets of n entities for consistency is exponential on the number n. Simpler forms of consistency are *2-consistency or arc consistency*, (i.e., checking for asserted relations between all pairs of individuals for disjoint relations) and *3-way consistency or path consistency* (i.e., checking all triples of individuals for inconsistencies caused by asserted relations and compositions of pairs of relations holding between the 3 entities).

There are cases where, *k-consistency* for a specific value of k implies strong *n-consistency* so that, a polynomial algorithm that enforces *k-consistency* also solves the *n-consistency* problem [93]. There also also cases where, although *k-consistency* does not imply *n-consistency*, there are specific sets of relations R_t (which are subsets of the set of all possible disjunctions of basic relations R), with the following property: if asserted relations are restricting to this set then *k-consistency* implies *n-consistency* and R_t is a *tractable set* of relations or a *tractable subset of* R [93].

Tractable subsets for point algebra have been identified in [110, 114, 112]. However, these results can be applied when point algebra is used for reasoning over projections of points in the 2D (or 3D) space in addition to the temporal case which involves only one dimension. In fact, point algebra is only applicable on point projections on each axis. Tractable sets of Allen interval algebra have been identified in [83] and [68]. These results apply in cases where Allen relations are used for spatial reasoning, since projections of 2D objects define 1D intervals on each axis, as in the case of temporal Allen relations [6]. A survey is presented in [63].

Tractability of RCC8 subsets is analysed in [91] while, cone-shaped directional relations are examined in [92]. Combining points and intervals for temporal reasoning is analysed in [55] while, combined reasoning over intervals and their durations is discussed in [89]. Recent results for topological and temporal relations are presented in [20]. A survey is presented in [93].

2.4 Temporal Representation and Reasoning in the Semantic Web

The OWL-Time temporal ontology[11] describes the temporal content of Web pages and the temporal properties of Web services. Apart from language constructs for the representation of time in ontologies, there is still a need for mechanisms for the representation of the evolution of concepts (e.g., events) in time. This is related to the problem of the representation of time in temporal (relational and object oriented) databases. Existing methods are relying mostly on temporal Entity Relation (ER) models [40] taking into account valid time (i.e., time interval during which a relation holds), transaction time (i.e., time at which a database entry is updated)

[11] http://www.w3.org/TR/owl-time/

or both. Also time is represented by time instants, intervals or finite sets of intervals [62]. Related work for infinite and indefinite temporal information is presented at [61, 33]. However, representation of time in OWL differs because (a) OWL semantics are not equivalent to ER model semantics in Relational Databases (e.g., OWL adopts the *Open World Assumption* while DBs typically adopt the *Closed World Assumption*) and (b) relations in OWL syntax are restricted to binary ones in contrast to DBs. Representation of time in the Semantic Web can be achieved using *Temporal Description logics (TDLs)* [3], *Concrete domains* [71], *Reification*, *labeling of properties* [43, 24], *Versioning* [58], *named graphs* [109] and *4D-fluents* [115].

Temporal Description Logics (TDLs) [3, 72] extend standard description logics (DLs) that form the basis for semantic Web standards with additional constructs such as "always in the past", "sometime in the future". TDLs offer additional expressive capabilities over non temporal DLs and retain decidability (with an appropriate selection of allowable constructs) but they require extending OWL syntax and semantics with the additional temporal constructs (the same as property labelling introduced in [43]). Representing information concerning specific time points requires support for concrete domains, resulting to the proliferation of objects [3].

Concrete Domains [71] introduce datatypes and operators based on an underlying domain (such as decimal numbers). The concrete domains approach requires introducing additional datatypes and operators to OWL, while our work relies on existing OWL constructs. This is a basic design decision in our work. TOWL [37] is an approach combining 4D-fluents with concrete domains but did not support qualitative relations, path consistency checking (as this work does) and is not compatible with existing OWL editing, querying and reasoning tools (e.g., Protege, Pellet, SPARQL).

Temporal RDF [43] proposes extending RDF by labelling properties with the time interval they hold. This approach also requires extending the syntax and semantics of the standard RDF. Note that Temporal-RDF cannot express incomplete information by means of qualitative relations. Although, in [53], Temporal-RDF was enhanced with support for undefined intervals it does not provide the full expressiveness of SOWL. Temporal-RDF is combined with fuzzy logic in [107].

Temporal annotation of properties as proposed in [43] has been proposed for OWL representation in [80], enhanced with support for undefined intervals. Querying support for annotated properties is provided as well [70].

Versioning [58] suggests that the ontology has different versions (one per instance of time). When a change takes place, a new version is created. Versioning suffers from several disadvantages: (a) changes even on single attributes require that a new version of the ontology be created leading to information redundancy (b) searching for events occurred at time instances or during time intervals requires exhaustive searches in multiple versions of the ontology, (c) it is not clear how the relation between evolving classes is represented. Furthermore, ontology languages such as

OWL are based on binary relations (relations connecting two instances) with no time dimension regarding ontology versions.

Named Graphs [109] represent the temporal context of a property by inclusion of a triple representing the property in a named graph (i.e., a subgraph into the RDF graph of the ontology specified by a distinct name). The default (i.e., main) RDF graph contains definitions of interval start and end points for each named graph, so that a temporal property is represented by the start and end points corresponding to the temporal interval that the property holds. Named graphs are not part of the OWL specification[12] (i.e., there are not OWL constructs translated into named graphs) and they are not supported by OWL reasoners. In [109] a SPARQL based temporal query language is also introduced applying only on quantitative defined temporal intervals.

Reification is a general purpose technique for representing n-ary relations using a language such as OWL that permits only binary relations. Specifically, an n-ary relation is represented as a new object that has all the arguments of the n-ary relation as objects of properties. For example if the relation R holds between objects A and B at time t, this is expressed as *R(A,B,t)*. Furthermore, in OWL, using reification this is expressed as a new object with R, A, B and t being objects of properties. Figure 2.8 illustrates the relation *WorksFor(Employee, Company, TimeInterval)* representing the fact that an employee works for a company during a time interval. Using reification, the extra class "ReifiedRelation" is created having all the attributes of the relation as objects of properties. Reification suffers mainly from two disadvantages: (a) a new object is created whenever a temporal relation has to be represented (this problem is common to all approaches based on OWL) and (b) offers limited OWL reasoning capabilities [115] since relation R is represented as the object of a property, thus OWL semantics over properties (e.g., inverse properties) are no longer applicable (i.e., the properties of a relation are no longer associated directly with the relation itself). Examples of temporal representation based on reification (the reified temporal relations are named *Events* or *Actions*) are presented at [99, 26]. In [108] temporal representation is combined with application specific SWRL rules for representing clinical narratives.

Using an improved form of reification, the N-ary relations approach suggests representing an N-ary relation as two properties each related with a new object (rather than as the object of a property, as reification does). This approach requires only one additional object for every temporal relation, maintains property semantics but (compared to the 4D-fluents approach below) suffers from data redundancy in the case of inverse and symmetric properties (e.g., the inverse of a relation is added explicitly twice instead of once as in 4D-fluents). This is illustrated in Figure 2.10. In the case of transitive properties additional triples are introduced as well.

[12] http://www.w3.org/TR/owl2-syntax/

Furthermore, domains and ranges of properties have to be adjusted taking into account the class of intermediate objects representing the relation (for example the *worksfor* relation in no longer a relation having as object an individual of class *Company* and subject of class *Employee* as they are now related to the new object "TemporalEmployment").

Similarly to our proposed approach (see Chapter 4), property restrictions (e.g., cardinality constraints) cannot be expressed directly on properties and, subsequently, can't be identified by a reasoner as it is common in OWL ontologies. Instead, restriction checking on properties have to be implemented separately (on top of the ontology) with extra rules. Software in Java, instead of SWRL as in this work, handling a subset of restrictions over only quantitative intervals has been also developed in our laboratory [98].

A plug-in for the Protege editor supporting editing of N-ary based temporal ontologies is presented at [94]. A similar tool has been developed in our laboratory for the proposed representations, both for 4D-fluents [73] and N-ary relations [87].

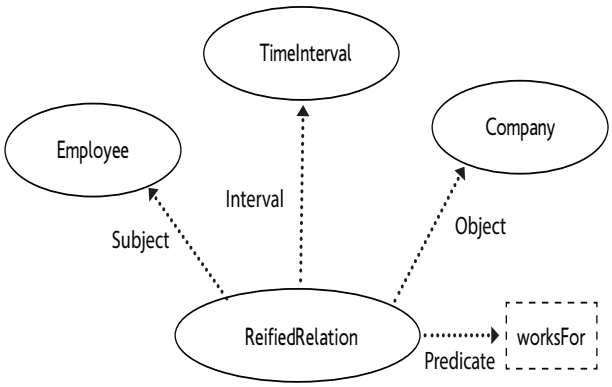

Figure 2.8: Example of Reification

The *4D-fluent* (perdurantist) approach [115] shows how temporal information and the evolution of temporal concepts can be represented in OWL. Concepts in time are represented as 4-dimensional objects with the 4th dimension being the time (*timeslices*). Time instances and time intervals are represented as instances of a *TimeInterval* class, which in turn is related with concepts varying in time as shown in Figure 2.9. Changes occur on the properties of the temporal part of the ontology keeping the entities of the static part unchanged. The 4D-fluent approach still suffers from proliferation of objects since it introduces two additional objects for each temporal relation (instead of one in the case of reification and N-ary relations). The N-ary relations approach referred to above is considered to be an alternative to the 4D-fluents approach considered into this work. Examples of representations

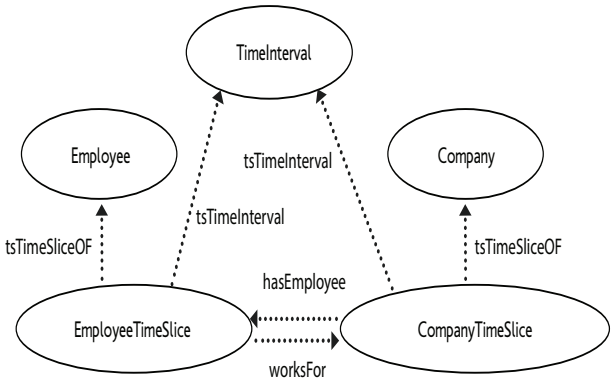

Figure 2.9: Example of 4D-fluents

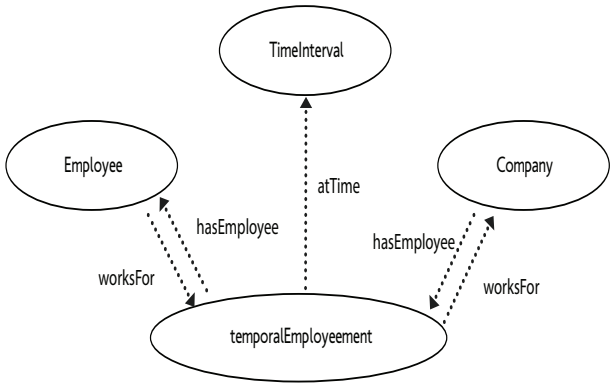

Figure 2.10: Example of N-ary Relations

based on 4D-fluents are presented at [14, 116, 9]. The MUSING system where both a 4D-fluents based approach and an alternative approach based on extending RDF with temporal annotation [67, 65, 66] used in conjunction with OWL-Time, but without the qualitative reasoning support proposed in this thesis, is a related work.

2.5 Spatial Representation and Reasoning in the Semantic Web

Formal spatial, and spatio-temporal representations have been studied extensively in the Database [45, 97] and recently, in the Semantic Web literature [23]. Spatial entities (e.g., objects, regions) in classic database systems are typically represented using points, lines (polygonal lines) or Minimum Bounding Rectangles *(MBRs)* enclosing objects or regions and their relationships [86]. Relations among spatial entities can be topological, orientation or distance relations. Furthermore, spatial

relations are distinguished into qualitative (i.e., relations described using lexical terms such as "Into", "South" etc.) and quantitative (i.e., relations described using numerical values such as "10Km away", "45 degrees North" etc.). Accordingly, spatial ontologies are defined based upon a reference coordinate system in conjunction with a set of qualitative topological and direction relations (e.g., RCC8 relations). Reasoning rules for various relation sets have been proposed as well [32, 93].

Representing spatio-temporal knowledge has also motivated research within the Semantic Web community. Katz et.al. [57] proposed representing RCC8 relations as OWL-DL class axioms (instead of object properties as in [42]) but this approach has limited scalability as shown in [106]. Chen et.al. [27] and Sotnykova et.al. [105] proposed an integrated spatio-temporal representation which includes qualitative relations but without specialized spatio-temporal reasoning support. Perry et.al. [100] proposed a representation based on quantitative spatio-temporal data and Christodoulakis et.al. [29] proposed a quantitative representation for data from GPS devices. Patkos et.al [85] proposed a representation combined with application specific rules based on Event calculus similar to the work presented at [8]. A representation for quantitative spatio-temporal information based on linear constraints is presented in [64, 60].

Pellet Spatial [106] offers reasoning support for RCC8 topological relations. In SOWL [11, 15, 13, 12] support for topological and directional (quantitative and qualitative defined) spatial relations is provided in conjunction with the temporal representation mechanism. Applications of the SOWL model for dynamic medical information[13], video content and spatial descriptions using qualitative terms are presented at [74], [46] and [30] respectively.

2.6 Querying Spatio-Temporal Information in the Semantic Web

Examples of temporal query languages for temporal databases include TQuel [104], TSQL2 [59] and ATSQL [21]. Query languages for RDF and OWL ontological representations are of particular interest as they form the basis for developing the new type of temporal ontology query languages. SeRQL [22] and SPARQL are good representatives of this category of query languages. SeRQL is a RDF/RDFS query language combining features of other (query) languages (e.g., RQL [56], RDQL [96], N-Triples, N3). Important features of SPARQL (and SeRQL) are: Graph transformation, RDF and XML Schema data type support, expressive path expression syntax and optional path matching. SPARQL (and SeRQL) supports comparison between date times.

[13] Available at: http://www.intelligence.tuc.gr/HPV-4dcase/

Query languages for RDF and OWL ontological representations such as SPARQL and SeRQL [22] form the basis for developing languages for querying spatio-temporal information in ontologies and the semantic Web. Querying spatio-temporal information over the semantic Web using languages such as SPARQL is a tedious task. Recent work on query languages for temporal ontologies include TOQL [8], t-SPARQL [109] and T-SPARQL [38] using 4D-fluents, named graphs and versioning respectively for the representation of temporal information.

2.6.1 t-SPARQL

t-SPARQL [109] extends the syntax of SPARQL with additional operators for expressing temporal queries. The data model used in this work, called temporal RDF, is based on named graphs and considers time as an additional dimension in data preserving the semantics of time. For the representation of time, OWL-Time has been extended with a new date format type called IntegerTime to express non-calendar time representations such as version numbers.

The language has two major usage formats: time point queries and temporal queries. Time point queries aim at retrieving information valid at a specified point in time. The "FROM SNAPSHOT t" expression signals the query engine to evaluate the querys graph pattern only on graph-elements valid at the time point "t", where "t" has to be a literal time value (the literal type is depending on the underlying time format). t-SPARQL queries, similarly to SOWL QL, are translated to SPARQL prior to execution. An example of a timepoint query is shown in figure 2.11

```
SELECT ?person FROM SNAPSHOT 1995
WHERE {
 ?person a foaf:Person .
}
```

Figure 2.11: t-SPARQL timepoint query example

Temporal queries allow the usage of wild card intervals and time points. These wild cards can be used to bind a variable to the validity period of a triple or to express temporal relationships between between intervals. t-SPARQL allows one form of temporal wildcards [?s,?e] which binds the literal start and end values.

An example of a timepoint query is shown in figure 2.12

```
SELECT ?s ?person
WHERE {
  [?s ,?e] ?person a foaf:Person .
  }
```

Figure 2.12: t-SPARQL temporal query example

SOWL QL currently supports all the features described above and, in addition, supports:

- A wider arsenal of temporal operators (quantitative and qualitative) such as the ALWAYS_AT and SOMETIME_AT temporal operators.
- A wide selection of spatial operators (quantitative and qualitative topological and directional operators) which are not supported by t-SPARQL.
- Reasoning for both temporal and spatial queries (a feature that is not supported by t-SPARQL).

2.6.2 T-SPARQL

T-SPARQL [38] is a TSQL2-Like Temporal Query Language for RDF. In T-SPARQL time is considered to be discrete, with a minimal system dependent representation unit called "chronon". A mono-temporal chronon corresponds to an elementary interval on the time axis, whereas a multi-temporal chronon corresponds to a unit hypercube in the N-dimensional time domain. Three base temporal types are defined for TSQL2 at the conceptual level: datetime, period and interval. The first one corresponds to an instantaneous event, without duration, which can be conventionally represented via a single chronon. The second one corresponds to a set of consecutive chronons along the time axis and is characterized by two datetime constants representing its boundaries. The third temporal type corresponds to a pure duration, non anchored on the time axis, and can be represented as a multiple of the chronon. The language T-SPARQL is equipped with the basic temporal constructs which have been designed for the TSQL2 relational query language and work with an extended set of temporal datatypes, functions and operators already present in the SPARQL specification.

T-SPARQL adds temporal selection capabilities to the SPARQL language by extending the basic graph pattern in the WHERE clause of the SELECT statement. As RDF triples are correspondingly augmented with the timestamp in the data model, graph patterns to be used in the T-SPARQL WHERE clause are extended

with an optional fourth position where matching with the triple timestamps can be specified. For example, in the graph pattern _:e ex:Dept "Toys" — ?t the variable ?t binds to the timestamp of a temporal triple representing the fact that an employee denoted by the blank node _:e has been working in the Toys department. The general syntax pattern is: (s, p, o — t). If the fourth position (along with the "—" separator) in the pattern is not used, that is a standard SPARQL three-position pattern is used, the matching with a temporal triple is made regardless of its timestamp. Also, the FILTER clause can be used, as in TSQL2, with the same semantics to specify constraints over timestamp variables.

Snapshot queries are used to extract a single temporal version from a multi version RDF graph. For instance, if the temporal RDF database encodes the definition of a multi-version ontology, the result of a snapshot query is a standard (non-temporal) RDF graph, which can be interpreted as a consistent single ontology version valid at a given time point.

SOWL QL currently supports all the features described above and, in addition, supports:

- A wider arsenal of temporal operators (quantitative and qualitative) such as the ALWAYS_AT and SOMETIME_AT temporal operators.
- A wide selection of spatial operators (quantitative and qualitative topological and directional operators) which are not supported by T-SPARQL.
- Reasoning support for both temporal and spatial queries (a feature that is not supported by T-SPARQL).

2.6.3 stSPARQL

stSPARQL extends SPARQL for querying stRDF. The model stRDF is a constraint data model that extends RDF with the ability to represent temporal and spatial data. This extension to RDF, follows the main ideas of constraint databases and represents tempora and spatial objects as quantifier-free formulas in a first-order logic of linear constraints. stSPARQL extends SPARQL with functions that take as arguments temporal and spatial terms and can be used in the SELECT, FILTER, and HAVING clause of a SPARQL query. A spatial term is a spatial literal (i.e., a typed literal with datatype strdf:geometry), a query variable that can be bound to a spatial literal, the result of a set operation on spatio-temporal literals (e.g., union), or the result of a geometric operation on spatial terms (e.g., buffer). sTSPARQL uses vectors of points to represent the different geometries and it does not support querying using qualitative operators.

SOWL QL has two main advantages over stSPARQL:

- Supports qualitative operators.
- Supports reasoning for both temporal and spatial queries (a feature that is not supported by sTSPARQL).

2.6.4 TOQL

TOQL [8] (Temporal Ontology Query Language) is an SQL-like language for OWL, supporting the basic structure of SQL (SELECT - FROM - WHERE) and treats classes and properties of an ontology almost like tables and columns of a database. TOQL supports queries over quantitative and qualitative spatio-temporal information in 4D-fluent ontologies. Nevertheless, TOQL syntax is independent of the underlying ontology representation mechanism.

TOQL not only uses SQL-like clauses and a similar syntax, but also treats ontologies almost like relational databases. Tables representing concepts correspond to classes and tables representing relations correspond to object properties. Attributes correspond to datatype properties. In addition, 1:1 and 1:N relations correspond to object properties. Table 2.1 summarizes the mapping between database relations and ontology concepts used by TOQL.

Relational Database	**Ontology**
Table representing concept	Class
Table representing N:N relation	Object Property
1:N or 1:1 relation	Object Property
Attribute	Datatype Property

Table 2.1: Mapping between database relations and ontology concepts.

In TOQL, the implementation of ALLEN operators correspond to comparisons between fluent properties. Fluent properties connect time slices and time slices are associated with time intervals. Consequently, the implementation of Allen operators correspond to comparisons between time intervals. The following operators are supported in TOQL: BEFORE, AFTER, MEETS, METBY, OVERLAPS, OVERLAPPEDBY, DURING, CONTAINS, STARTS, STARTEDBY, ENDS, ENDEDBY and EQUALS, representing the corresponding relations holding between two time intervals.

The following TOQL query retrieves the name of the company that hired employee “x” and *then* employee “y”:

SELECT Company.companyName
FROM Company, Employee AS E1, Employee AS E2
WHERE Company.hasEmployee:E1 BEFORE Company.hasEmployee:E2
AND E1.employeeName like "x" AND E1.employeeName LIKE "y"

TOQL also introduces clause "AT" which compares a fluent property (i.e., the time interval in which the property is true) with a time period (time interval) or time point:

- **AT(time point)** operation returns true if the time interval holds true at the time specified.
- **AT(start time point, end time point)** operation returns true if the time interval holds true for *all* the time interval.

The following TOQL query retrieves the name of the company employee "x" was working for, from time=3 to time=5:

SELECT Company.companyName
FROM Company, Employee
WHERE Company.hasEmployee:Employee AT(3,5)
AND Employee.employeeName LIKE "x"

Because TOQL is independent of the mechanism implementing time, there is no way to directly access class *TimeInterval* (i.e., the class holding values of time). In order for TOQL to return values of time, the keyword TIME is introduced. It follows datatype or object properties and can be used only in SELECT. It returns the start and end time point (if any) in which the property holds true (the time interval in which the property is true). If no end point exists, it returns only its start point. As an example, the following TOQL query retrieves the time for which a company had employee "x":

SELECT Company.hasEmployee.TIME
FROM Company, Employee
WHERE Company.hasEmployee:Employee AND
Employee.employeeName LIKE "x"

In addition to the existing set of temporal operators (i.e. the AT and Allen operators) the language is enhanced with spatial operators for handling both, spatial and temporal relations, thus the IN_RANGE and all RCC8 and directional relations are supported by corresponding operators.

Spatial operators refer to topological or directional spatial relations represented in the underlying ontology. The result of applying spatial operators are locations qualifying the expressions specified by the query. Locations are assessed using their names (although the user can issue queries addressing the underlying quantitative representation using coordinates, but formulating such queries requires that the user be familiar with the underlying spatio-temporal representation). The following spatial operators are supported:

- NORTH_OF
- NORTHEAST_OF
- EAST_OF
- SOUTH_OF
- WEST_OF
- SOUTHWEST_OF
- SOUTHEAST_OF
- INTO
- OUTSIDE_OF
- SAME_LOCATION_AS
- BORDERING
- OVERLAPPING
- CONTAINS
- INTERNALLY_BORDERING
- CONTAINS_AND_BORDERING
- IN_RANGE.

They correspond to the eight directional relations in Fig. 2.6, the RCC8 relations in Fig. 2.5 and one operator (range) involving distance information. Spatial operators are issued in WHERE, followed by a string denoting the location name according to the pattern <SPATIAL OPERATOR> <STRING>. For example the following query retrieves the name of the company located north of a given location:

SELECT Company.companyName
FROM Company
WHERE Company NORTH_OF "Attica"

Queries involving the IN_RANGE operator have the following syntax: IN_RANGE <Comparison operator> <Number> OFF <String> where the string denotes location name. The following query retrieves the names of employees working for companies located in distance greater than 100Km away from "Athens":

SELECT Employee.employeeName
FROM Company, Employee
WHERE Company IN_RANGE >100 OFF "Athens" AND
Company.hasEmployee:Employee

All spatial and temporal operators can be combined in TOQL in order to form spatio-temporal queries. For example the following query retrieves the name of the company located *north* of a given location *at* a specific instant of time:

SELECT Company.companyName
FROM Company
WHERE Company NORTH_OF "Attica" AT(5)

Finaly, compared to SOWL QL, TOQL:

- Is not a semantic Web approach but is intended to be used by those familiar with relational databases.
- Does not support queries with time points that do not exist in the ontology. In TOQL, all time points used as arguments in temporal operators should be explicitly declared in the knowledge base before querying. A query involving time points that are not asserted into the ontology will return the empty set as a result. SOWL QL can handle this case by inserting the data specified by the query into the knowledge base and by invoking the reasoner prior to query processing.
- Has only two quantitative operators: The AT(timepoint) and the AT(timepoint,timepoint) operators. SOWL QL supports a more powerful set of quantitative temporal operators including all ALLEN operators used as quantitative(in TOQL ALLEN operators are only used as qualitative).

- Does not support any spatial quantitative operators like SOWL QL's Point(x,y) operator.

Chapter 3

The SOWL Model

SOWL 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 [115] (or, equivalently, on the N-ary [84]) approach. Both RCC8 topological 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 "north- 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. Both, the 4D-fuents and the N-ary relations approaches are expanded to accommodate this information. The SOWL reasoner implements path consistency [93], and is capable of inferring new relations and checking their consistency, while retaining soundness, completeness, and tractability over the supported sets of relations.

3.1 SOWL Model

Temporal representation in SOWL is based on the 4D-fluents approach enhanced with Allen relations which are defined as object properties between intervals. Topological and directional spatial relations are represented as object properties defining the relation between the spatial extends of objects (which can be static or moving). In each case, the spatial representation is combined with the temporal representation with the location of objects being a static or fluent property respectively. An alternative implementation based on the N-ary approach has been implemented as well.

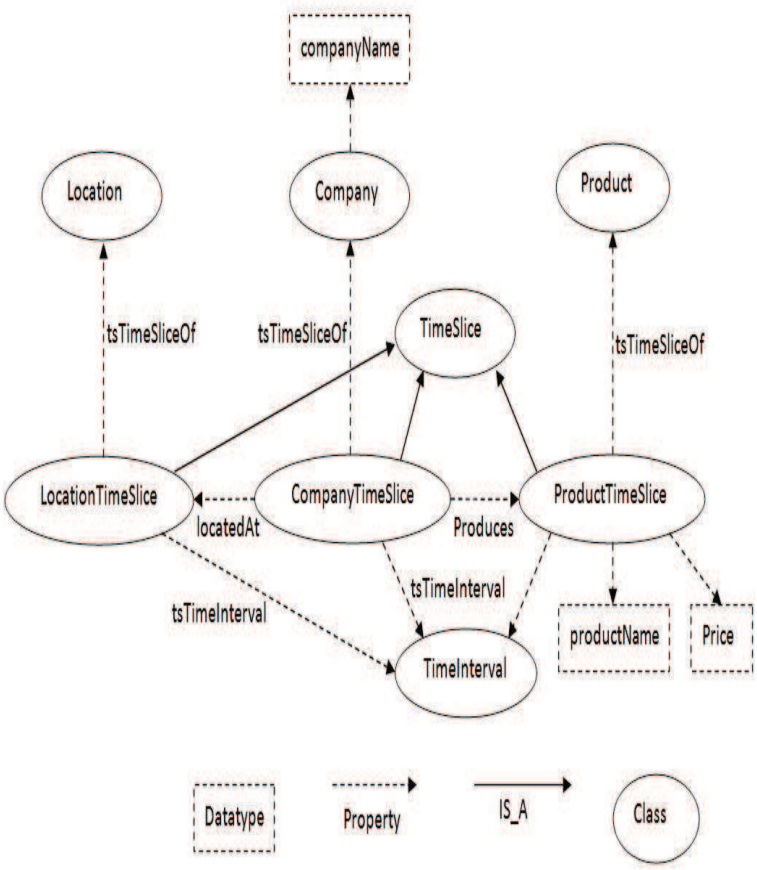

Figure 3.1: Dynamic Enterprise Ontology

3.1.1 Temporal Representation using 4D-fluents

Following the approach by Welty and Fikes [115], to add time dimension to an ontology, classes *TimeSlice* and *TimeInterval* with properties *tsTimeSliceOf* and *tsTimeInterval* are introduced. Class *TimeSlice* is the domain class for entities representing temporal parts (i.e., "time slices") and class *TimeInterval* is the domain class of time intervals. A time interval holds the temporal information of a time slice. Property *tsTimeSliceOf* connects an instance of class *TimeSlice* with an entity, and property *tsTimeInterval* connects an instance of class *TimeSlice* with an instance of class *TimeInterval*. Properties having a time dimension are called fluent properties and connect instances of class *TimeSlice*.

Figure 3.1 illustrates a temporal ontology with classes *Company* (with datatype property *companyName*), *Product* (with datatype properties *price* and *product-*

Name), and *Location* which represents spatial information (see Figure 3.2 and Figure 3.3). In this example, *CompanyName* is static property (its value does not change in time), while properties *produces*, *productName*, *locatedAt* and *price* are dynamic (fluent) properties whose values may change in time. Because they are fluent properties, their domain (and range) is of class *TimeSlice*. *CompanyTimeSlice*, *LocationTimeslice* and *ProductTimeSlice* are instances of class *TimeSlice* and are provided to denote that the domain of properties *produces*, *locatedAt*, *productName* and *price* are time slices restricted to be slices of a specific class. For example, the domain of property *productName* is not class *TimeSlice* but it is restricted to instances that are time slices of class *Product*. All fluent properties are defined as *subproperties* of the property $fluent$.

In SOWL, the 4D-fluent representation is enhanced with qualitative temporal relations holding between time intervals whose starting and ending points are not specified. This is implemented by introducing temporal relationships as object relations between time intervals. This can be one of the 13 pairwise disjoint Allen's relations [1] of Figure 2.4.

By allowing for qualitative relations the expressive power of the representation increases. Typically, the 4D-fluents model (similarly to other approaches such as Temporal RDF [43]), assume closed temporal intervals for the representation of temporal information, while semi-closed and open intervals can not be represented effectively in a formal way. In SOWL, this is handled by Allen relations: for example if interval t_1 is known and t_2 is unknown but we know that t_2 starts when t_1 ends, then we can assert that t_2 is *met by* t_1. Likewise, if t_3 is an interval with unknown endpoints and t_3 is $before$ t_1 then, using compositions of Allen relations [1], we infer that t_3 is $before$ t_2 although both interval's endpoints are unknown and their relation is not represented explicitly in the ontology. Semi-closed intervals can be handled in a similar way. For example, if t_1 starts at time point 1, still holds at time point 2, but it's endpoint is unknown, we assert that t_1 has *started by* interval t_2:[1,2].

SOWL demonstrates enhanced expressiveness compared to previous approaches [109, 105, 23, 27, 43, 37, 100, 54] by combining 4D-fluents with Allen's temporal relations, their formal semantics and composition rules as defined in [1]. Notice that, temporal instants still cannot be expressed; subsequently, relations between time instants or between instants and intervals cannot be expressed explicitly.

In this work, an instant-based (or point-based) approach is adopted. Definitions for temporal entities (e.g., instants and intervals) are provided by incorporating OWL-Time into the same ontology. Each interval (which is an individual of the $ProperInterval$ class) is related with two temporal instants (individuals of the $Instant$ class) that specify it's starting and ending points using the *hasBegining* and *hasEnd* object properties respectively. In turn, each $Instant$ can be related with a specific date using the concrete *dateTime* datatype.

One of the $before$, $after$ or $equals$ relations may hold between any two temporal instants with the obvious interpretation. In fact, only the relation $before$ is needed since relation $after$ is defined as the inverse of $before$ and relation $equals$ can be represented using the $sameAs$ OWL keyword applied on temporal instants. In this work, for readability we use all three relations. Notice also that, property $before$ may be also qualitative when holding between time instants or intervals whose values or end points are not specified. This way, we can assert and infer facts beyond the ones allowed when only instants or intervals with known values (e.g., dates) or endpoints are allowed. Quantitative defined instants are specified using the $dateTime$ datatype and the supported operators can be applied between instants.

Relations between intervals are expressed as relations between their starting and ending points, which, in turn are expressed as a function of the three possible relations between points (time instants) namely $equals$, $before$ and $after$ denoted by "=", "<" and ">" respectively, forming the so called "point algebra" [111]. Let $i_1 = [s_1, e_1]$ and $i_2 = [s_2, e_2]$ be two intervals with starting and ending points s_1, s_2 *and* e_1,e_2 respectively; then, the 13 Allen relations of Fig. 2.4 are rewritten as follows:

$$
\begin{aligned}
&i_1\ before\ i_2 \equiv e_1 < s_2\\
&i_1\ equals\ i_2 \equiv s_1 = s_2 \wedge e_1 = e_2\\
&i_1\ overlaps\ i_2 \equiv s_1 < s_2 \wedge e_1 < e_2 \wedge e_1 > s_2\\
&i_1\ meets\ i_2 \equiv e_1 = s_2\\
&i_1\ during\ i_2 \equiv s_1 > s_2 \wedge e_1 < e_2\\
&i_1\ starts\ i_2 \equiv s_1 = s_2 \wedge e_1 < e_2\\
&i_1\ finishes\ i_2 \equiv s_1 > s_2 \wedge e_1 = e_2
\end{aligned}
$$

The relations *after, overlappedby, metby, contains, startedby and finishedby* are the inverse of *before, overlaps, meets, during, starts and finishes* and are defined accordingly (by interchanging s_1, s_2 and e_1, e_2 in their respective definitions). Notice that, in the case of Allen relations additional relations (representing disjunctions of basic relations) are introduced in order to implement path consistency, totalling a set of 29 supported relations (although, such relations are not required by a point algebra). Example of such relations is the disjunction of relations *during, overlaps* and *starts*. The full set of supported relations is presented in Appendix A. These temporal relations and the corresponding reasoning mechanism are integrated within the SOWL ontology.

In the original work by Welty and Fikes [115], the following restriction is imposed on timeslices: whenever two timeslices are related by means of a fluent property, their corresponding temporal intervals must be equal. However, no mechanism for enforcing this restriction is provided. In this work, the following SWRL rule

in conjunction with the reasoning mechanism of Chapter 4 imposes the required restriction:

$$fluent(x, y) \wedge tsTimeInterval(y, z) \wedge tsTimeInterval(x, w) \rightarrow equals(w, z)$$

3.1.2 Temporal Representation using N-ary Relations

The N-ary version of the SOWL ontology introduces one additional object for representing a temporal property. This object is an individual of class *Event* and this name convention is also adopted by other approaches such as the LODE ontology [99]. In SOWL, the temporal property remains a property relating the additional object with both the objects (e.g., an *Employee* and a *Company*) involved in a temporal relation. This is illustrated in Figure 2.10. The representation of qualitative relations between temporal intervals or points (and the corresponding reasoning mechanisms) remain identical to the 4D-fluents based version of the model.

The advantage of the N-ary approach over reification [99] is that property semantics are retained. For example, when a property is the inverse of another, the inverse property declaration is retained. As shown in Figure 2.10, if *worksFor* is the inverse of *hasEmployee* and *Employee*1 is related with *Company*1 using the *worksFor* relation and an intermediate *EmploymentEvent*1, OWL semantics indicate that the relation *hasEmployee* holding between *Company*1 and *Employee*1 through the *EmploymentEvent*1 object can be inferred. This is not the case with reification because, as shown in Figure 2.8, the *worksFor* relation is the object of a property, and objects of properties do not have inverses as the properties themselves. The same hold for symmetric and reflexive properties. Transitive properties are more involved since the equality of the related intervals must also hold when a transitive property applies. This can be achieved using an SWRL rule such as in the case of 4D-fluents.

N-ary relations (similarly to 4D-fluents) require modification of domains and ranges of fluent properties. Specifically, when a property is temporal, if the domain of property is *ClassA* and the range is *ClassB* (where domains and ranges can be composite class definitions or atomic concepts), then using the N-ary representation the domain becomes *ClassA OR Event* and the range *ClassB OR Event*. Compared to 4D-fluents, the disjunction of concepts appearing both in domain and ranges of properties limits specificity of references of the N-ary representation.

3.1.3 Spatial Representation

The 4D-fluent mechanism is also enhanced with several types of qualitative spatial relations. These can be either topological or directional [93]. Figure 3.2 illustrates a general ontology representation model for spatial information. Class *Location* has

attribute *name* (of type string). Also a *Location* object can be optionally connected with a *Geometry* class with subclasses: *Point, Line, Polyline and MBR*. Class *Point* has two (or three in a three-dimensional representation) numerical attributes, namely X, Y (also Z in a three-dimensional representation). For example, *Point* will be the *Geometry* of entities such as cities in a large scale map. Class *Line* has *point1* and *point2* as attributes representing the ending points of a line segment. Class *PolyLine* represents the surrounding contour of an object (or region) as a set of consecutive line segments.

An object (or region) may also be represented by its *Minimum Bounding Rectangle* (MBR) specified by the four numerical attributes *Xmax, Ymax, Xmin and Ymin*. Both representations may co-exist in SOWL (using one of them or both is a design decision).

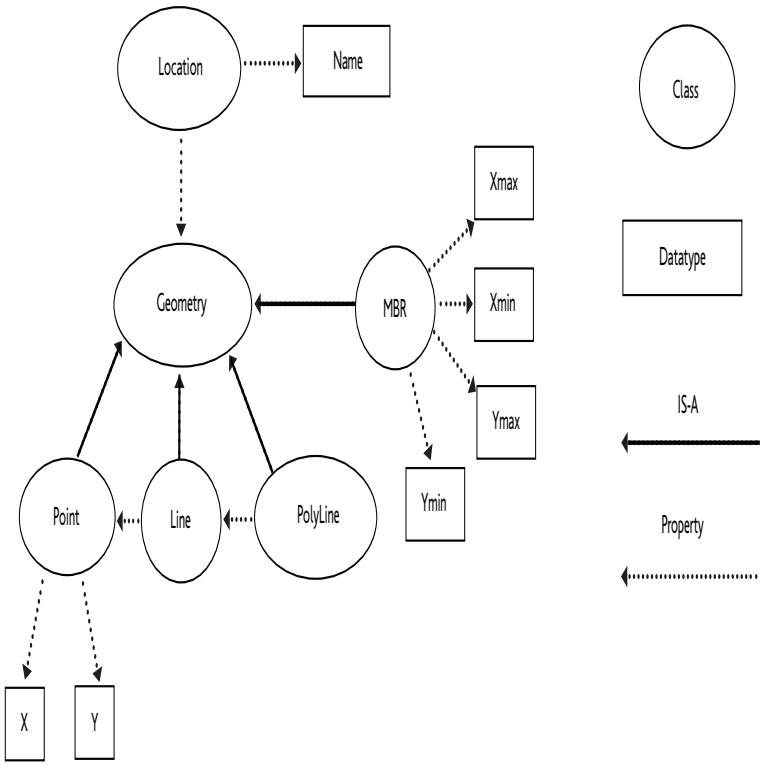

Figure 3.2: Ontology representation of spatial objects.

The spatial relations between regions can be easily extracted from their surrounding *MBRs* (or contours) by comparing their coordinates. In the case of MBR or point-based representations, extraction of qualitative relations from the underlying quantitative representations has been implemented with SWRL rules and embedded into the ontology as part of the reasoning mechanism. In the case of polygons, a separate software component is used for extracting qualitative relations [46].

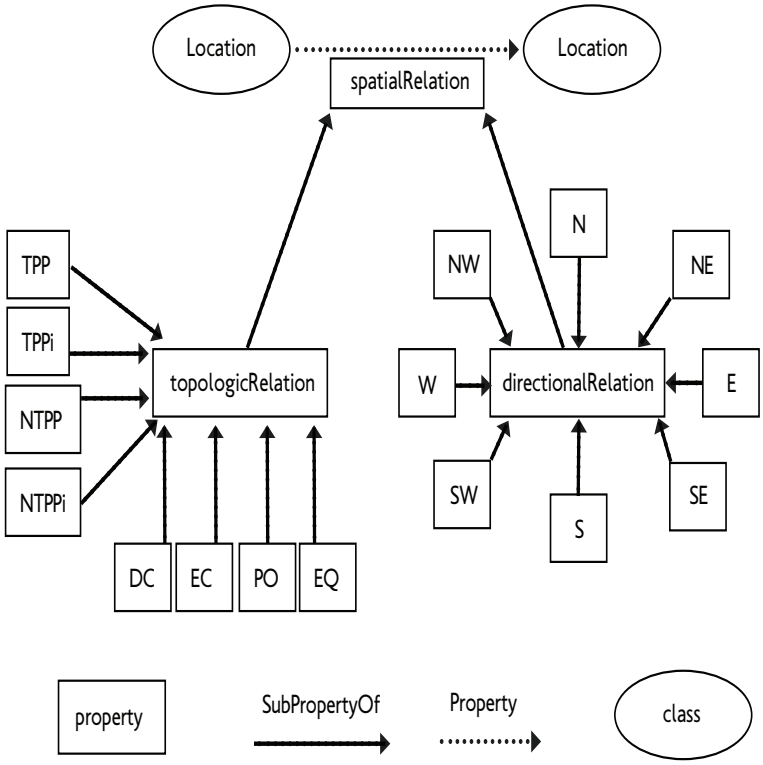

Figure 3.3: Ontology schema with spatial relations.

In an ontology, each *spatialRelation* connects two locations and has two subproperties namely: *topologicRelation* and *directionalRelation*. Figure 3.3 summarizes all types of spatial relations within a common ontology schema. Omitting one or more types of spatial relations is also a design decision.

The topologic relations shown in Figure 2.5, *(DC, EC, EQ, NTTP, NTTPi, TTP, TPPi, PO)*, referred to as RCC8 relations [90], are also defined in SOWL. In order to implement a sound and complete reasoning mechanism additional relations are are introduced totalling a minimal set of 49 relations (See Chapter 4). Direction relations are defined based on cone-shaped areas [36]. Other alternative approaches based on 2D-projections are presented at [93, 103]. As shown in Figure 2.6, eight direction relations can be identified namely *North (N), North East (NE), East (E), South East (SE), South (S), South West (SW), West (W) and North West (NW)* following the cone-shaped areas approach of [36].

The cone shaped approach is suitable for objects represented by points (e.g., by their centroid). It complements topological relations that apply on regions. Nevertheless, for completeness, a projection-based approach has been implemented as well. When applied to points, the projection based approach is equivalent to applying point algebra on a pair of orthogonal axes (instead of one in the temporal

case), while representing regions corresponds to applying Allen's interval algebra on two axes (one to each dimension) instead of one as in the temporal case. The projections over the horizontal axis define relations *East* and *West* (equivalently *left* and *right* relations) corresponding to the *Before* and *After* relations respectively of the temporal representation. The projections over the vertical axis define the relations *North* and *South*, (equivalently *front* and *behind* relations).

In contrast to the cone-shaped approach, each axis is independent and different relations (thus and their conjunction) may hold as long as they are defined over a different axis (e.g., *North* and *South* are disjoint properties). Thus, relations *Noth*, *West* and consequently the relation *North-West* may hold simultaneously. This is not the case in the cone-shaped approach where all basic properties are pairwise disjoint. Notice also that, although the cone-shaped and the projection based directional relations result in the same relations set, they convey different semantics and consequently call for different reasoning mechanism. Selecting one of the two approaches referred to above (i.e, cone-shaped or project-based) is subject to user preference or may depend on the application at hand. For example, one may opt for the projection-based approach in the case of large objects or the cone-shaped approach in the case of small objects or objects specified by their coordinates. Both approaches are implemented in SOWL. If the cone-shaped approach is selected, a set of additional disjunctive relations are required and they are part of the representation as well. In case of the projection based approach the additional relations are not required (See Chapter 4).

Finally, distance relations are defined and can be used in conjunction with the above relation types. Notice that, qualitative distance relations (e.g., *far* and *near*) may be ambiguous especially in applications where a common scale for measuring distances is not provided. This is resolved when distance relations are expressed quantitatively (e.g., 3Km away from city A) and stored in the ontology as N-ary relations [84] (i.e., by defining an object with attributes the two related locations and a numerical attribute representing their distance). In SOWL, we opt for the later (quantitative) approach for representing distance information.

3.1.4 Combining Spatial and Temporal Representations

In the case of a moving object, its location is a property of a timeslice holding for a specific time interval (Figure 3.5) while, in the case of a static object, its location is a property of the object and not a property of a timeslice.

Notice that, even if the location of the object is static, some of its properties may change in time so that, there can be timeslices associated with it (e.g., timeslices of a building for different owners in time). In the N-ary based approach, the location of a moving object is a property of the *Event* object that has specific temporal extends. In the case of a static object, its location remains a property of the object. As in

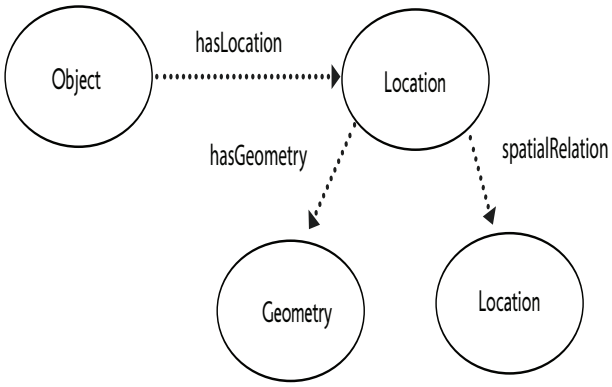

Figure 3.4: Ontology representation of static objects.

4D-fluents, the object can have temporal properties represented by *Event* objects while its location is a static property. Notice that, in case of locations changing continuously [44], trajectory parameters can be represented instead of locations. Reasoning and querying support for such representation is a direction of future work.

Figure 3.6 illustrates the dynamic ontology schema representing the scenario "T1 Radio was produced in Patras, a city west of Athens from May 2006 to May 2010, since then it is produced at Athens". In this example, we do not know whether the product is still produced in Athens. Only the first temporal interval is defined. The second interval and both locations are unknown and only qualitative relations about them appear into the ontology.

The example of Fig. 3.6 illustrates the applicability of the model in the case of missing or inaccurate information (as it is usually the case with natural language descriptions). In these cases, models based on quantitative information only, are insufficient.

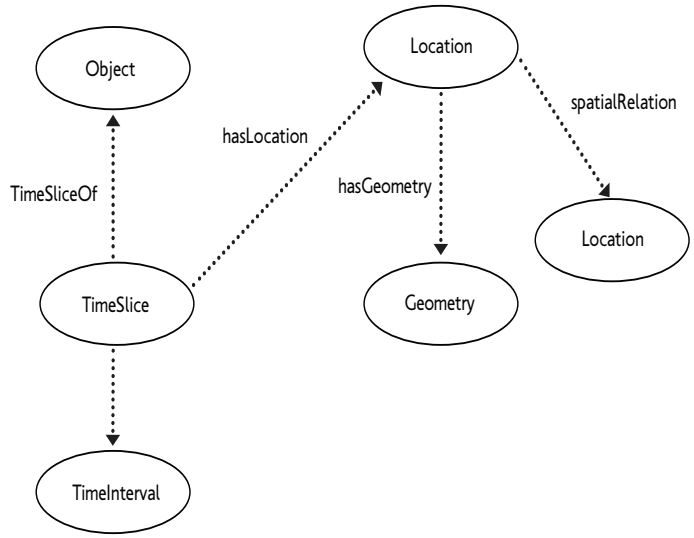

Figure 3.5: Ontology representation of moving objects.

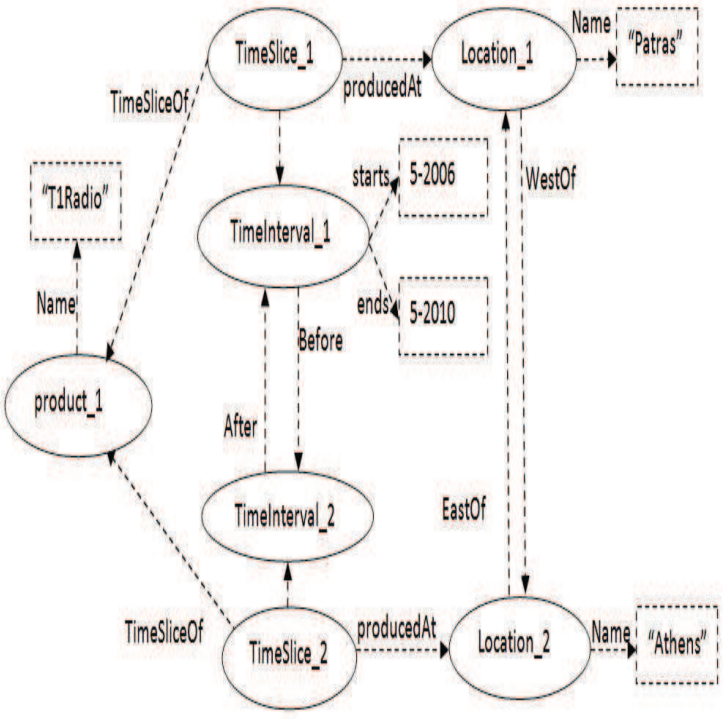

Figure 3.6: Instantiation example.

Chapter 4

Reasoning in SOWL

Temporal and spatial reasoning in SOWL is realized by introducing a set of SWRL[1] rules operating on spatial (topological or directional) relations as well as, by a set SWRL rules for asserting inferred temporal Allen relations. Reasoners that support DL-safe rules (i.e., rules that apply only on named individuals in the knowledge base) such as Pellet [102] can be used for inference and consistency checking over spatio-temporal relations. Alternatively, OWL axioms on temporal properties can be used instead of SWRL. However, this approach cannot guarantee decidability and is therefore not compatible with W3C specifications. In addition to reasoning applying on temporal and spatial relations presented in Section 4.1 and Section 4.2 respectively , the Pellet reasoner applies to the ontology schema for inferring additional facts using OWL semantics (e.g., facts due to symmetric relationships and class-subclass relationships). Checking for property restrictions on temporal properties (*fluent* properties) is also implemented and discussed in Section 4.3.

4.1 Temporal Reasoning

Reasoning is applied either on temporal intervals directly [15] or by applying point-based reasoning [17] operating on representations of intervals involving their starting and ending points. Both approaches have been implemented and are discussed in the following.

4.1.1 Temporal Reasoning over Interval-Based Representations

Reasoning is realized by introducing a set of SWRL rules operating on temporal intervals. The temporal reasoning rules are based on the composition of pairs

[1] http://www.w3.org/Submission/SWRL/

of the basic Allen's relations of Figure 2.4 as defined in [1]. The composition table of basic Allen's relations is presented in Table 4.1. Relations BEFORE, AFTER, MEETS, METBY, OVERLAPS, OVERLAPPEDBY, DURING, CONTAINS, STARTS, STARTEDBY, ENDS, ENDEDBY and EQUALS are represented using symbols B, A, M, Mi, O, Oi, D, Di, S, Si, F, Fi and = respectively. Compositions with EQUALS are not presented since these compositions keep the initial relations unchanged.

	B	A	D	Di	O	Oi	M	Mi	S	Si	F	Fi
B	B	B,A,D,Di,O,Oi M,Mi,S,Si,F,Fi, Eq	B,O,M, D,S	B	B	B,O,M,D,S	B	B,O,M,D, S	B	B	B,O,M, D,S	B
A	B,A,D,Di, O,Oi,M, Mi,S,Si,F, Fi,Eq	A	A,Oi,Mi D,F	A	A,Oi, Mi,D,F	A	A,Oi,Mi, D,F	A	A,Oi,Mi D,F	A	A	A
D	B	A	D	B,A,D,Di,O,Oi M,Mi,S,Si,F,Fi, Eq	B,O,M, D,S	A,Oi,Mi,D,F	B	A	D	A,Oi,Mi D,F	D	B,O,M, D,S
Di	B,O,M,Di, Fi	A,Oi,Di,Mi,Si	O,Oi,D, Di,S,Si, F,Fi,Eq	Di	O,Di,Fi	Oi,Di,Si	O,Di,Fi	Oi,Di,Si	O,Di,Fi	Di	Oi,Di,Si	Di
O	B	A,Oi,Di,Mi,Si	O,D,S	B,O,M,Di,Fi	B,O,M	O,Oi,D,Di,S,Si, F,Fi,Eq	B	Oi,Di,Si	O	O,Di,Fi	O,D,S	B,O,M
Oi	B,O,M,Di, Fi	A	Oi,D,F	A,Oi,Di,Mi,Si	O,Oi,D, Di,S,Si, F,Fi,Eq	A,Oi,Mi	O,Di,Fi	A	Oi,D,F	Oi,A,Mi	Oi	Oi,Di,Si
M	B	A,Oi,Di,Mi,Si	O,D,S	B	B	O,D,S	B	F,Fi,Eq	M	M	O,D,S	B
Mi	B,O,M,Di, Fi	A	Oi,D,F	A	Oi,D,F	A	S,Si,Eq	A	Oi,D,F	A	Mi	Mi
S	B	A	D	B,O,M,Di,Fi	B,O,M	Oi,D,F	B	Mi	S	S,Si,Eq	D	B,O,M
Si	B,O,M,Di, Fi	A	Oi,D,F	Di	O,Di,Fi	Oi	O,Di,Fi	Mi	S,Si,Eq	Si	Oi	Di
F	B	A	D	A,Oi,Di,Mi,Si	O,D,S	A,Oi,Mi	M	A	D	A,Oi,Mi	F	F,Fi,Eq
Fi	B	A,Oi,Di,Mi,Si	O,D,S	Di	O	Oi,Di,Si	M	Oi,Di,Si	O	Di	F,Fi,Eq	Fi

Table 4.1: Composition table for Allen's temporal relations.

The composition table represents the result of the composition of two Allen

relations. For example, if relation R_1 holds between $interval_1$ and $interval_2$ and relation R_2 holds between $interval_2$ and $interval_3$ then, the entry of Table 4.1 corresponding to row R_1 and column R_2 denotes the possible relation(s) holding between $interval_1$ and $interval_3$. Not all compositions yield a unique relation as a result. For example, the composition of relations *During* and *Meets* yields the relation *Before* as a result while, the composition of relations *Overlaps* and *During* yields three possible relations namely *Starts, Overlaps and During*. Rules corresponding to compositions of relations R_1, R_2 yielding a unique relation R_3 as a result can be represented using SWRL as follows:

$$R_1(x, y) \wedge R_2(y, z) \rightarrow R_3(x, z) \tag{4.1}$$

An example of temporal inference rule is the following:

$$DURING(x, y) \wedge MEETS(y, z) \rightarrow BEFORE(x, z)$$

Rules yielding a set of possible relations cannot be represented in SWRL since, disjunctions of atomic formulas are not permitted as a rule head. Instead, disjunctions of relations are represented using new relations whose compositions must also be defined and asserted into the knowledge base. For example, the composition of relations *Overlaps* and *During* yields the disjunction of three possible relations (*During, Overlaps* and *Starts*) as a result:

$$OVERLAPS(x, y) \wedge DURING(y, z) \rightarrow$$

$$During \vee Starts \vee Overlaps$$

If the relation *DOS* represents the disjunction of relations *During, Overlaps* and *Starts*, then the composition of *Overlaps* and *During* can be represented using SWRL as follows:

$$OVERLAPS(x, y) \wedge DURING(y, z) \rightarrow DOS(x, z)$$

The set of possible disjunctions over all basic Allen's relations contains 2^{13} relations, and reasoning over all temporal Allen relations has exponential time complexity [63]. However, tractable subsets of this set that are closed under composition (i.e., compositions of relation pairs from this subset yield also a relation in this subset) are also known to exist [83, 112]. In addition, inverse axioms (relations AFTER, METBY, OVERLAPPEDBY, STARTEDBY, CONTAINS and FINISHEDBY are the inverse of BEFORE, MEETS, OVERLAPS, STARTS, DURING and FINISHES respectively) and rules defining the relation holding between two intervals with known starting and ending points (e.g., if the ending point of $interval_1$ is before the starting point of $interval_2$ then, $interval_1$ is *before* $interval_2$) are also asserted into the knowledge base.

The starting and ending points of intervals are represented using concrete datatypes such as *xsd:date* that support ordering relations. Axioms involving disjunctions of basic relations are denoted using the corresponding axioms for these basic relations. Specifically, compositions of disjunctions of basic relations are defined as the disjunction of the compositions of these basic relations. For example, the composition of relation *DOS* (representing the disjunction of *During, Overlaps and Starts*), and the relation *During* yields the relation *DOS* as a result as follows:

$$DOS \circ During \rightarrow (During \vee Overlaps \vee Starts) \circ During \rightarrow$$

$$(During \circ During) \vee (Overlaps \circ During) \vee (Starts \circ During)$$

$$\rightarrow (During) \vee (During \vee Overlaps \vee Starts) \vee (During)$$

$$\rightarrow During \vee Starts \vee Overlaps \rightarrow DOS$$

The symbol $\circ$ denotes composition of relations. Compositions of basic (non-disjunctive) relations are defined using Table 4.1. Similarly, the inverse of a disjunction of basic relations is the disjunction of the inverses of these basic relations illustrated in Figure 2.4. For example, the inverse of the disjunction of relations $Before$ and $Meets$ is the disjunction of their inverse relations, $After$ and $MetBy$ respectively.

By applying compositions of relations, the implied relations may be inconsistent (i.e., yield the empty relation $\perp$ as a result). Consistency checking is achieved by applying path consistency [93, 83, 112]. Path consistency is implemented by consecutive application of the formula:

$$\forall x, y, k \; R_s(x, y) \leftarrow R_i(x, y) \cap (R_j(x, k) \circ R_k(k, y)) \tag{4.2}$$

representing intersection of compositions of relations with existing relations. Symbol $\cap$ denotes intersection, symbol $\circ$ denotes composition and symbols R_i, R_j, R_k, R_s denote Allen relations. The formula is applied until a fixed point is reached (i.e., application of rules does not yield new inferences) or until the empty set is reached, implying that the ontology is inconsistent.

An additional set of rules defining the result of intersection of relations holding between two intervals is also introduced. These rules are of the form:

$$R_1(x, y) \wedge R_2(x, y) \rightarrow R_3(x, y), \tag{4.3}$$

where R_3 can be the empty relation. For example, the intersection of relation *DOS* (represents the disjunction of *During, Overlaps and Starts*) with relation *During*, yields relation *During* as a result:

$$DOS(x, y) \wedge During(x, y) \rightarrow During(x, y).$$

The intersection of relations *During* and *Starts* yields the empty relation, and an inconsistency is detected:

$$Starts(x, y) \wedge During(x, y) \rightarrow \bot.$$

The maximal tractable subset of Allen relations containing all basic relations when applying path consistency comprises of 868 relations [83]. Tractable subsets of Allen relations containing 83 or 188 relations [112] can be used instead, offering reduced expressiveness but increased efficiency over the maximal subset of [83]. Furthermore, since the proposed temporal reasoning mechanism affects only relations of temporal intervals, it can be also applied to other temporal representation methods (besides 4D-fluents) such as N-ary relations. Reasoning operating on temporal instants rather on intervals is also feasible [112]. Specifically, qualitative relations involving instants form a tractable set if relation $\neq$ (i.e., a temporal instant is before *or* after another instant) is excluded. Reasoning involving relations between interval and instants is achieved by translating relations between intervals to relations between their endpoints [1].

Path consistency requires composition of properties, intersection of properties and role complement. Notice that, disjointness of properties can be represented in terms of complement of properties (i.e., two properties are disjoint when one of them is *subproperty* of the *complement* of the second property). However, the combination of property composition, intersection and complement has been proven to be undecidable [95]. Instead of property complement, the disjointness of two properties can be represented as an *at most 0* cardinality constraint over their intersection. However, the intersection and the composition of two properties is a composite (i.e., not simple) property and applying cardinality constraints over composite properties has been proven to be undecidable [52]. Therefore, reasoning using SWRL, as proposed in this thesis, is the only solution complying with current OWL specifications while retaining decidability.

Implementing path consistency over Allen relations (or topological and directional spatial relations) requires minimizing the required additional relations and rules for implementing the mechanism. Existing work (e.g., [91]) emphasizes on determining maximal tractable subsets of relations while, practical implementations calls for minimizing of such relation sets (i.e., finding the minimal tractable set that contain the required relations). For example, implementing path consistency over the maximal tractable set of Allen relations [91], containing 868 relations is impractical, since defining all intersections and compositions of pairs of relations by means of SWRL rules requires millions of such rules.

In this work, minimal relation sets containing a tractable set of basic relations are detected by applying the *closure method* of Table 4.2 (i.e., starting with a set of relations, intersections and compositions of relations are applied iteratively until no

new relations are produced). Since compositions and intersections are constant-time operations (i.e., a bounded number of table lookup operations at the corresponding composition tables) the running time of closure method is linear to the total number of relations of the identified tractable set. Applying the closure method over the set of basic Allen relations yields a tractable set containing 29 relations.

```
Input: Set S of tractable relations
 Table C of compositions
 WHILE S size changes
      BEGIN
           Compute C:Set of compositions of relations in S
           S=S ∪ C
           Compute I:set of intersections of relations in S
           S= S ∪ I
      END
 RETURN S
```

Table 4.2: Closure method

. Implementing path consistency over point algebra does not require introducing additions relations besides the basic ones.

Notice that, implementing path consistency using rules of the form of Equation 4.2 over n relations requires $O(n^3)$ rules (i.e., rules for every possible selection of three relations must be defined), while implementing path consistency using rules according to Equation 4.1 and Equation 4.3 (as implemented in this work) requires $O(n^2)$ rules, since rules for every pair of relations must be defined. Further improvements and reductions can be achieved by observing that the disjunction of all basic Allen relations when composed with other relations yields the same relation, while intersections yield the other relation. Specifically, given that *All* represents the disjunction of all basic relations and, R_x is a relation in the supported set then the following hold for every R_x:

$$All(x, y) \wedge R_x(x, y) \rightarrow R_x(x, y)$$
$$All(x, y) \wedge R_x(y, z) \rightarrow All(x, z)$$
$$R_x(x, y) \wedge All(y, z) \rightarrow All(x, z)$$

Since relation *All* always holds between two individuals, because it is the disjunction of all possible relations, all rules involving this relation, both compositions and intersections, do not add new relations into the ontology and they can be safely removed. Also, all rules yielding the relation *All* as a result of the composition of two supported relations R_{x1}, R_{x2}:

$$R_{x1}(x, y) \wedge R_{x2}(y, z) \rightarrow All(x, z)$$

can be removed too. Thus, since intersections yield existing relations and the fact that the disjunction over all basic relations must hold between two intervals, all rules involving the disjunction of all basic relations and consequently all rules yielding this relation can be safely removed from the knowledge base. After applying this optimization the required number of axioms for implementing path consistency over the minimal tractable set of Allen relations is reduced to 983.

4.1.2 Reasoning over Point-Based Representations

The possible relations between temporal instants are $before$, $after$ and $equals$, denoted as "<",">","=" respectively. Table 4.3 illustrates the set of reasoning rules defined on the composition of existing relation pairs.

Relations	$<$	$=$	$>$
$<$	$<$	$<$	$<,=,>$
$=$	$<$	$=$	$>$
$>$	$<,=,>$	$>$	$>$

Table 4.3: Composition Table for point-based temporal relations.

The composition table represents the result of the composition of two temporal relations. For example, if relation R_1 holds between $instant_1$ and $instant_2$ and relation R_2 holds between $instant_2$ and $instant_3$ then, the entry of Table 4.3 corresponding to row R_1 and column R_2 denotes the possible relation(s) holding between $instant_1$ and $instant_3$. Also, the three temporal relations are declared as pairwise disjoint, since they cannot simultaneously hold between two instants. Not all compositions yield a unique relation as a result. For example, the composition of relations $before$ and $after$ yields all possible relations as a result. Because such compositions do not yield new information these rules are discarded. Rules corresponding to compositions of relations R_1 *and* R_2 yielding a unique relation R_3 as a result are retained (7 out of the 9 entries of Table 4.3 are retained) and are expressed in SWRL using rules of the form (Equation 4.1):

$$R_1(x, y) \wedge R_2(y, z) \rightarrow R_3(x, z)$$

The following is an example of such a temporal inference rule:

$$before(x, y) \wedge equals(y, z) \rightarrow before(x, z)$$

Therefore, 7 out of the 9 entries in Table 4.1 can be expressed using SWRL rules while, the two remaining entries do not convey new information. A series of compositions of relations may imply relations which are inconsistent with existing ones.

Consistency checking is achieved by imposing path consistency [112]. Path consistency is implemented by iteratively applying formula of Equation 4.2:

$$\forall x, y, k \; R_s(x, y) \leftarrow R_i(x, y) \cap (R_j(x, k) \circ R_k(k, y))$$

representing intersection of compositions of relations with existing relations (symbol $\cap$ denotes intersection, symbol $\circ$ denotes composition and R_i, R_j, R_k, R_s denote temporal relations). The formula is applied until a fixed point is reached (i.e., the consecutive application of the rules above does not yield new inferences) or until the empty set is reached, implying that the ontology is inconsistent. In addition to rules implementing compositions of temporal relations, a set of rules defining the result of intersecting relations holding between two instances must also be defined in order to implement path consistency. These rules are of the form of Equation 4.3:

$$R_1(x, y) \wedge R_2(x, y) \rightarrow R_3(x, y)$$

where R_3 can be the empty relation. For example, the intersection of the relation representing the disjunction of *before, after and equals* (abbreviated as ALL), and the relation $before$ yields the relation $before$ as result:

$$ALL(x, y) \wedge before(x, y) \rightarrow before(x, y)$$

The intersection of relations $before$ and $after$ yields the empty relation, and an inconsistency is detected:

$$before(x, y) \wedge after(x, y) \rightarrow \bot$$

As shown in Table 4.3, compositions of relations may yield one of the following four relations: *before, after, equals* and the disjunction of these three relations. Intersecting the disjunction of all three relations with any of these leaves existing relations unchanged. Intersecting any one of the tree basic (non disjunctive) relations with itself also leaves existing relations unaffected. Only compositions of pairs of different basic relations affect the ontology by yielding the empty relation as a result, thus detecting an inconsistency. By declaring the three basic relations *before, after, equals* as pairwise disjoint, all intersections that can affect the ontology are defined. Path consistency is implemented by defining compositions of relations using SWRL rules and by declaring the three basic relations as disjoint. Notice that, path consistency is sound and complete when applied on the three basic relations [111].

Alternatively, we can define the composition of $before$ with itself as a transitivity axiom rather than by an SWRL rule. In this case, there would be no need for SWRL rules applying only on named individuals into the ontology ABox. The resulting representation will apply on the TBox as well. However, this is not compatible with OWL 2.0 thus imposing the use of SWRL rules: relation $before$ must

be declared as transitive in order to infer implied relations and disjoint with $after$, its inverse relation, (also $before$ is asymmetric and irreflexive) in order to detect inconsistencies. However, OWL specifications[2] disallow the combination of transitivity and disjointness (or asymmetry) axioms on a property since they can lead to undecidability [49]. This restriction is necessary in order to guarantee decidability of the basic reasoning problems for OWL 2 DL.

In cases where temporal information is provided as dates, the qualitative relations are specified using SWRL rules that apply on the quantitative representation. An example of such a rule is the following:

$$Instant(x) \wedge Instant(z) \wedge inXSDDateTime(x, y)$$
$$\wedge inXSDDateTime(z, w) \wedge lessThan(y, w) \rightarrow before(z, x)$$

Replacing the $lessThan$ operator in the rule with $greaterThan$ and $equal$ yields the corresponding rules for relations $after$ and $equals$ respectively. These qualitative relations can be combined with asserted and inferred qualitative relations using path consistency.

All interval relations can be represented by means of point relations between their end-points. Rules implementing transformation of Allen relations to endpoint relations and rules yielding Allen relations from endpoint relations have been implemented as well. For example, the rule yielding the *During* Allen relation from endpoint relations is the following:

$$ProperInterval(a) \wedge ProperInterval(x) \wedge before(b, y)$$
$$\wedge before(z, c) \wedge hasBeginning(a, b)$$
$$\wedge hasBeginning(x, y) \wedge hasEnd(a, c) \wedge hasEnd(x, z) \rightarrow intervalDuring(x, a)$$

Rules similar to the above, yielding all basic Allen relations are implemented. Notice that, the inverse transformation cannot be expressed by a single SWRL rule: one Allen relation corresponds to four end-point relations and conjunctions at the rule head are not supported in SWRL. Conjunctions can be expressed as rules with identical antecedent part and different head. For example, the following rules represent the transformation of relation $IntervalOverlaps$:

$$hasBeginning(a, b) \wedge hasBeginning(x, y) \wedge hasEnd(a, c) \wedge hasEnd(x, z)$$
$$\wedge intervalOverlaps(x, a) \rightarrow before(z, c)$$

$$hasBeginning(a, b) \wedge hasBeginning(x, y) \wedge hasEnd(a, c) \wedge hasEnd(x, z)$$

[2] http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/#The_Restrictions_on_the_Axiom_Closure

$$\wedge intervalOverlaps(x, a) \rightarrow before(b, z)$$

$$hasBeginning(a, b) \wedge hasBeginning(x, y) \wedge hasEnd(a, c) \wedge hasEnd(x, z)$$

$$\wedge intervalOverlaps(x, a) \rightarrow before(y, b)$$

$$hasBeginning(a, b) \wedge hasBeginning(x, y) \wedge hasEnd(a, c) \wedge hasEnd(x, z)$$

$$\wedge intervalOverlaps(x, a) \rightarrow before(y, c)$$

In fact, the last rule in the above example is implied by the previous rules and the rules that specify that the start of an interval is before the end and it can be omitted.

Notice that, if data consistency can be assured, then reasoning can be significantly speeded-up. In cases where all relations are specified quantitatively (i.e., by numerical values) reasoning with path consistency can be dropped. For example, for intervals with known end-points, all possible relations between them can be computed in quadratic time from their end-point dates. The computed set of relations is guaranteed to be consistent and reasoning is not needed.

If consistency checking is not needed (in case instance assertions does not contain conflicts -implied or direct) then, temporal properties need not be declared disjoint. For example if sequences of events are recorded using sensors, then there is a valid arrangement of the events on the axis of time (i.e., the sequence of their recording), thus their temporal relations are consistent by definition. In this case, reasoning can be achieved using OWL role inclusion axioms instead of SWRL rules that apply on the ontology TBox as well. Such axioms are of the form:

$$before \circ equals \sqsubset before$$

All relation compositions can be defined similarly. Intersections of relations are not required in case of basic point algebra relations and if the consistency checking requirement is dropped, only OWL axioms are sufficient for implementing the reasoning mechanism. In this case, a great speed up is achieved, since in our experiments for over 20,000 random instances, reasoning is achieved in about 18 seconds (which is comparable to the time required for reasoning over 80 instances using SWRL when consistency checking is required). This speed-up can be achieved only in special cases where consistency of data is guaranteed (thus consistency checking can be dropped), which is not the case for example in natural language text.

4.2 Spatial Reasoning

In the following, reasoning over both, topological and directional relations is discussed. Choosing either representation is a design decision that depends mainly on the application. However, both representations may co-exist in the SOWL model (both are common in natural language expressions and may co-exist e.g., in text descriptions over the Web). A third case, is reasoning over interval-based (or their equivalent point-based representations) obtained as the projections of spatial entities (i.e., points or regions in a two-dimensional or three-dimension space). Reasoning over interval-based (or point-based) spatial projections is equivalent to reasoning over temporal intervals (or points) discussed in Section 4.1.

Table 4.5 illustrates a composition table for RCC8 topological relations [15]. The corresponding composition table for directional relations is illustrated in Table 4.4. Spatial reasoning is then achieved by applying rules implementing the inferred relations of the composition table at hand.

As shown in Table 4.5, only a limited set of table entries leads to an unambiguous result. For example, the composition of the *NTPP* and *DC* topologic relations (i.e., object A is into B and object B outside of C) yields the *DC* relation as a result meaning that A is outside of C. However, the composition of *NTPP* and *PO* relations does not yield a unique relation as a result. Only 27 out of the 64 (RCC-8) entries of Table 4.5, and only 8 out of the 64 compositions of basic cone-shaped directional relations [36] can be used to infer unique relations.

The SOWL spatial representation implements reasoning rules for RCC8 relations and cone-shaped direction relations using SWRL and OWL 2.0 property axioms. All basic relations are pairwise disjoint. Their inverse relations (e.g., North is the inverse of South) are defined as well. Furthermore, the point identity relation (O) is handled using the OWL *SameAs* keyword applied on points instead of explicitly asserting the relation. 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 [31, 36, 92]. The supported directional relations are the 9 basic relations and their disjunctions appearing in Table 4.4. Compositions and intersections of disjunctive relations are defined using the compositions and intersections of basic relations as in the case of temporal reasoning.

The directional relations in SOWL (under the assumption that the line separating two 2D cone-shaped areas e.g., North from North-West, is part of only one of these areas, preserving the disjointness of basic relations) are a special case of the revised Star Calculus [92] and is decided by path consistency when applied to basic relations. Furthermore, given a tractable set of relations, by applying compositions, intersections and inverse operations until a set of relations that is closed under these operations is yielded, the resulting relations set is also tractable [93]. By applying this *closure method* to the basic relations of Figure 2.6 a tractable set of relations

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

Table 4.4: Composition table for cone-shaped directional relations.

containing the basic directional relations and all relations appearing in Table 4.4 is yielded. This set of directional relations is used in this work for directional spatial reasoning.

Reasoning on RCC8 relations also combines OWL property axioms along with a set of composition rules (i.e., rules defining compositions of RCC8 relations) and intersection rules. Specifically, relations *DC, EC* and *PO* are symmetric, and relations *NTPPi* and *TPPi* are inverse of *NTPP* and *TPP* respectively. *EQ* corresponds to the equality relation for *Location* objects. In SOWL, the spatial reasoner implements the RCC8 composition rules of Table 4.5. For example the rule defining the

	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 4.5: Composition table for RCC8 topological relations.

composition of relations $TPPi$ and $NTPPi$ for locations x, y, z is the following:

$$TPPi(x, y) \wedge NTPPi(y, z) \rightarrow NTPPi(x, z)$$

Extracting spatial relations from the raw spatial data depends on the application and is not part of the reasoning mechanism, besides the specific case of MBRs where rules for extracting both projection based directional relations and RCC8 relations given the MBRs coordinates have been implemented. Extracting directional and topologic applications from random polygons has been implemented in our laboratory as an external application [46].

Notice that, using the full set of relations (totalling $2^8 - 1$ relations in case of

RCC8) leads to intractability since this set is not decided by path consistency. However, tractable subsets of the full set are known to exist [93, 91]. Such subsets are used in this work offering increased expressive power while retaining tractability. Specifically applying the closure method of Table 4.2 over RCC8 topological and cone-shaped directional relations yields two sets with 49 and 33 relations respectively. Implementing path consistency over these sets as described in Section 4.1.1 requires a total of 1439 and 964 axioms respectively [15].

4.2.1 Reasoning over Point-Based Spatial Representations using Point Algebra

The point based representation and reasoning method presented in Section 4.1.2 applies also to spatial data in 2 (or 3) dimensions with points represented by their x, y (and z) coordinates. Then, relations $East$ and $West$ between regions or locations are defined using their point projections on X axis, relations $South$ and $North$ are defined using projections on Y axis while, relations $Bellow$ and $Above$ are defined using projections on Z axis. The reasoning mechanism for point based temporal information applies in the case of spatial information as it is, by replacing the temporal relations with the corresponding spatial relations on each axis (i.e., $North$ and $South$, $East$ and $West$, $Bellow$ and $Above$ replace relation $before$ and $after$ on the X, Y and Z axis respectively). Rules defining relations representing conjunctions of basic relations (e.g., $NorthWest$ is the conjunction of $North$ and $West$)are also defined:

$$North(x, y) \wedge West(x, y) \rightarrow NorthWest(x, y)$$

and the inverse:

$$NorthWest(x, y) \rightarrow North(x, y)$$

$$NorthWest(x, y) \rightarrow West(x, y)$$

Adding equivalent rules for relations $NorthEast$, $SouthWest$ and $SouthEast$ along with rules extracting qualitative relations by comparing point coordinates, are sufficient for realizing a projection based spatial reasoning mechanism for directional relations. Notice that, the reasoning mechanisms referred to above is based on projections and is not compatible with reasoning over cone-shaped relations. In the later case relations such as $NorthWest$ are not the conjunction of relations such as $North$ and $West$ as in the case of projections. Reasoning for cone-shaped relations is discussed in Section Section 4.2. Whether a projection based or a cone-shaped approach is adopted is a design decision.

In case of regions represented by their Minimum Bounding Rectangles (MBRs), the point based approach still applies. Two SWRL rules (one for each axis) are introduced for computing the coordinates of the centroid (i.e., as the average of the maximum and minimum values over each axis). Then, the directional relations between regions are defined as the directional relations between their centroids. Finally, topological relations can be also extracted by comparing the coordinates of an MBR representation.

4.3 Restriction Checking over Temporal Properties

Checking for restrictions holding on time dependent (fluent) properties requires particular attention. If a fluent property holds between two objects (classes), then, these objects are only indirectly associated through one or more artificial objects (e.g., TimeSlice object in 4D-fluents). Notice that both, reification and the 4D-fluents approach introduce additional objects for expressing fluent properties. A fluent property is declared between the artificial object and an actual object (as in Figure 2.10 and 2.8) or between two artificial objects (as in Figure 2.9). Checking for property restrictions would require adjusting the domain and range of this property from the artificial to the actual objects (e.g., to *Company* and *Employee* objects in Figure 2.10). This, in turn, calls for extra rules or software, which is a disadvantage pertaining to all methods considered in this work (i.e., 4D-fluents and N-ary relations). For example, for the *worksfor* property in Figure 2.9, the domain of the property is no longer class *Employee* but *timeslice of Employee*. Accordingly, its range is *timeslice of Company*.

Similar adjustments must be made in the case of N-ary relations but, in this case, combining transitivity of properties while retaining domain and range restrictions becomes problematic: for example, the $worksfor$ relation in Figure 2.10 must be provided with two alternative domains and ranges. Other restrictions on properties such as symmetric, asymmetric, reflexive, irreflexive and transitive can be applied directly on the temporal property retaining the intended semantics.

Universal restrictions (e.g., "all Employees work for a company") also require adjusting domains and ranges (i.e., all timeslices of employees $workfor$ timeslices of companies). Existential restrictions are adjusted as well (if for example each employee must work for some company then, timeslices of employees must work for some timeslices of companies). Notice that, an existential restriction corresponds to an *at least one* qualified cardinality restriction in OWL and the way it is handled is discussed in the rest of this section.

Adjusting cardinality restrictions, functional and inverse functional properties is

somewhat more complicated. Functional properties are a special case of cardinality restrictions (i.e., if a property is functional, then each object must be connected to *at most* one subject which is different for each object). Cardinality restrictions are amenable to two different interpretations depending on the specific application and the intended semantics: Cardinality restrictions may be interpreted, either as restricting the total number of individuals of a class (e.g., *Company*) related with each individual of another class (e.g., *Employee*) through a fluent property at all times or, as restricting the number of individuals for each specific temporal interval that the fluent property holds true. The first interpretation is handled simply by counting on the number of individuals of a class related with this property and is implemented in SWRL (because OWL cardinality restrictions cannot handle fluent properties connecting objects through intermediate objects).

The following rule expresses the restriction that each employee can work for at most n companies. If $n+1$ company individuals are found to connect with an employee individual, then the restriction is violated (the $Alldifferent$ keyword is an abbreviation for a series of axioms imposing that the n+1 individuals $z_1, z_2 ... z_{n+1}$ are all different). By imposing a *max* cardinality restriction of 0 over property *error* at the definition of class *Employee* the violation of the cardinality restriction is detected by standard reasoners such as Pellet using the rule:

$$(At-most-rule1)Employee(x) \wedge (tsTimesliceOf(x_1, x)$$

$$\wedge ... \wedge tsTimesliceOf(x_{n+1}, x)$$

$$\wedge worksfor(x_1, y_1) \wedge worksfor(x_{n+1}, y_{n+1}$$

$$\wedge tsTimesliceOf(y_1, z_1) ... \wedge tsTimesliceOf(y_{n+1}, z_{n+1})$$

$$\wedge Alldifferent(z_1, z_2, ..., z_{n+1})$$

$$\wedge Company(z_1) ... \rightarrow error(x, z_1)$$

An $at-least$ restriction is expressed similarly as follows: an *at most* $n-1$ rule is applied (changing the asserted property to $satisfies(x, n)$) followed by an *at least one* cardinality restriction on the $satisfies$ property for class *Employee*. An *exact* cardinality restriction can be expressed by combining an *at least n* with an *at most n* restriction. All rules impose also a restriction on the type of objects involved (e.g., they require that only company objects are involved by checking only for objects connected with timeslices of companies). Dropping such a check leads to an unqualified numeric restriction on the property. Notice that, the *Open World Assumption* of OWL will cause a reasoner (e.g., Pellet) not to detect an inconsistency of an $at-least$ restriction as future assertions might cause invalidity of this inconsistency detection. Instead, the the user can retrieve individuals that are not yet proven to satisfy the restriction using the following SPARQL query:

```
select distinct ?x
where {
?x rdf:type ex1:Employee.
OPTIONAL{
?x ex1:satisfies ?y.}
FILTER(!bound(?y))}
```

The second interpretation imposes restrictions on the number of individuals associated with an individual of a specific class through a fluent property for every temporal interval that the property holds true. Checking for such restrictions requires applying reasoning rules over the relations between the temporal intervals associated with the fluent property as described in Section 4.1. The next step is to detect overlapping and non-overlapping intervals. After the Allen relations holding between intervals have been inferred, Allen properties *during, contains, starts, startedby, finishes, finishedby, overlaps, overlapedby, equals* are defined as subproperties of property *overlapping*, thus detecting overlapping and non-overlapping intervals. Also, properties *before, after, meets, metby* are defined as subproperties of property *non-overlapping*.

Expressing an *at most* restriction for every time interval is based on the following observation: the restriction is violated iff $n+1$ distinct individuals are connected with a given individual (through their timeslices) with the relation at hand, and their corresponding intervals are all pairwise overlapping. Iff $n+1$ intervals are pairwise overlapping then, there exist an interval where $n+1$ intervals share a common sub-interval, and this can be proven by induction on n. The existence of such an interval implies that for this interval the *at least* restriction is violated. The corresponding rule (used in combination with a cardinality restriction on property error for inconsistency detection by reasoners) is expressed as (the *pairwiseoverlapping* is an abbreviation for a set of *overlapping* relations between all pairs of intervals at hand):

$$(At-most-rule2)Employee(x) \wedge (tsTimesliceOf(x_1, x)$$
$$\wedge ... \wedge tsTimesliceOf(x_{n+1}, x) \wedge hasinterval(x_{n+1}, w_{n+1})$$
$$\wedge worksfor(x_1, y_1) \wedge worksfor(x_{n+1}, y_{n+1}$$
$$\wedge tsTimesliceOf(y_1, z_1) ... \wedge tsTimesliceOf(y_{n+1}, z_{n+1}) \wedge$$
$$Alldifferent(z_1, ..., z_{n+1}) \wedge pairwiseoverlapping(w_1, ...w_{n+1})$$
$$\wedge Company(z_1) ... \rightarrow error(x, z_1)$$

The case of an *at least* restriction applying for every interval that a fluent property holds is handled as follows: every time instant related with an interval that

the fluent property in question holds, is also related with the object of the corresponding class. For example if a *worksFor* property holds for each *Employee* then all temporal instants that the property holds are detected by asserting an *OccuresAt* relation between the *Employee* and the time instants. There are three sub-properties of *OccuresAt* namely, *duringAt, endsAt* and *startsAt* indicating that the time instant is *during*, at the *start* or at the *end* of an interval that the property holds. For each such sub-property assertion, an SWRL rule is applied.

The second step is to check if for each time instant that the property *occuresAt* for an individual, the restriction at hand is satisfied. For example, in the case of an *at least 2* restriction applied on individuals of a class, a time instant that the fluent holds satisfies the restriction iff (a) the fluent *startsAt* this point and also another point that *equals* the point in question *startsAt* this fluent (b) the fluent *endsAt* the point and another point that *equals* the point in question *endsAt* this fluent (c) the fluent holds *duringAt* the point in question and also this point is *into* a second interval that the property holds or is *equals* both the end of such an interval and the beginning of another. Each case is implemented as an SWRL rule.

Each instant indicating the end or the start of an interval is a distinct individual from other points even if they represent the same time point (e.g., if a point is the end of an interval and the beginning of another then two points declared as *equal* must be asserted). Finally, as in the case of the *at least* restriction in the first interpretation, since an inconsistency cannot be detected by OWL reasoners, a SPARQL query is issued in order to detect individuals for which the restriction is not satisfied yet.

All rules under both interpretations involve a time consuming selection of all possible subsets of individuals and intervals. Therefore, expressing restrictions using SWRL may become time consuming. Specifically, rules for imposing a cardinality of *at-least* or *at most* n involves selection of all combinations of n among k timeslices (or reified relations) (where k is the number of temporal individuals in the ontology) it is not scalable for large values of n. In the case of reification or N-ary relations, cardinality constraints are expressed accordingly, using appropriate adjustments on classes and on properties of involved objects. To the best of our knowledge this is the only known solution to the problem of cardinality restriction checking on temporal representations.

Besides representation of cardinality and value restrictions and adjustments of domains and ranges the following object property characteristics are redefined using 4D-fluents as follows:

- *Functional*: It is handled as an at most 1 unqualified cardinality restriction.
- *Inverse Functional*: The inverse property is handled as an at most one unqualified cardinality restriction.

- *Symmetric*: The fluent property is *symmetric* too, thus the symmetry axioms applies on the interval that the involved timeslices exist.
- *Asymmetric*: This is handled as a form of cardinality restriction where the same property cannot hold for interchanged subjects and objects for timeslices that share an overlapping interval.
- *Equivalent*: The fluent properties are equivalent too.
- *Reflexive*: The fluent property is reflexive too, thus when a timeslice has the property for an interval it is also the subject of the property for this interval.
- *Irreflexive*: This is handled as an cardinality restriction; two timeslices of an object can not be related with the property in question if their intervals overlap.
- *Subproperty*: subproperty axioms apply for the fluent properties with the intended semantics.
- *Transitive*: Fluent properties can be safely declared transitive since related timeslices must have equal intervals (by the definition of the 4D-fluent model) and for these intervals the transitivity is applied.

Datatype properties have fewer characteristics (i.e., *subproperty, equivalence disjointness, functional*) and they are handled as is the case of object properties. In case of the N-ary relations the above adjustments must take into account the different objects involved (i.e., *Events* instead of *timeslices*).

Chapter 5

SOWL Query Language

Representing spatial and temporal information in OWL using mechanisms such as 4D-fluents or N-ary relations requires introducing additional objects that complicate the ontology. Querying spatio-temporal information in OWL using query languages such as SPARQL leads to complicated queries and requires that the users be familiar with the underlying representation mechanism. Adding spatial and temporal operators that hide the underlying representation from the end user is an important issue to deal with.

The main goal of spatio-temporal query languages is to maintain simplicity of expression. Desirable features of temporal query languages include, temporal upward compatibility (i.e., conventional queries and modifications on temporal relations still can be executed), point and interval-based views of data and ease of implementation.

In this chapter we present SOWL QL, a query language which extends SPARQL with temporal and spatial operators following the examples of [109]. SOWL QL is model independent and introduces a powerful set of spatio-temporal operators which allow users to query spatio-temporal OWL ontologies without dealing with the underlying model representation. Compared to the existing work, SOWL has the following three advantages: (a) supports both spatial and temporal operators, (b) it is capable of querying over both quantitative and qualitative spatio-temporal information and (c) supports reasoning during the querying process. The working version of SOWL QL[1] is implemented on top N-ary relations representation, although a representation based on 4D-fluents has been implemented as well. Notice that, SOWL QL queries are translated into equivalent SPARQL queries, thus SOWL Query Language can be considered as a form of "syntactic sugar" over SPARQL for spatio-temporal queries.

[1]The working version of SOWL QL can be downloaded from http://www.intelligence.tuc.gr

5.1 Language syntax and semantics

Being an extension of SPARQL, SOWL QL uses the same clauses SPARQL does and fully supports all SPARQL features. The structure of a SOWL QL query is the same as a SPARQL with the addition of SOWL QL operators. As we discussed in 2.1.4 the basic evaluation mechanism for SPARQL queries is based on graph matching where the query criteria are given in the form of *subject* - *predicate* - *object* RDF triples refered to as basic graph patterns. Basic graph patterns are used inside the body of the WHILE clause in a SPARQL query and this is where SOWL QL operators are applied. The Generic syntax of the language is shown in Figure 5.1. Below we show all available query patterns of SOWL QL explaining how the language deals with time and space. First we discuss the temporal query patterns and then the spatial. Finally, we show how these are combined to produce spatio-temporal query patterns.

```
SELECT Variable(s)
WHERE { Triple(s)  spatial or temporal operators
AND Condition(s)
}
```

Figure 5.1: SOWL QL Generic systax

Pattern 1:
Subject Predicate Object.

An ontology consists of two parts: the static part (classes, properties, instances) and the dynamic part consisting of additional temporal classes needed to represent time and their evolution in time as well as properties and instances of the above temporal classes (e.g., TimeSlice class and tsTimeSliceOf property for the 4D-fluents model or Event class and atTime property for the N-ary relations model). SOWL QL first determines if a property (object or datatype) in a query is a fluent property (i.e., a property that connects Timeslices or Events) or not (a property that connects "static" classes or a "static" class with a datatype). In the later case, the query is a static one and is handled as an ordinary SPARQL query. In a subject - predicate - object triple a temporal property is referred to as a "dynamic" predicate.

PATTERN 1:

Subject dynamicPredicate Object

Figure 5.2: Triple with dynamic predicate

As shown in Figure 5.2, the triple involving a dynamic predicate resembles a static one. task of the query translation is to determine if the predicate is dynamic or not by checking the ontology for dynamic objects connected to that predicate. More specifically, in cases where the predicate is a fluent, the predicate connects two dynamic objects (Timeslices or Events) rather than connecting two static concepts of the ontology (e.g a Company and an Employee).

Apart from retrieving dynamic objects when the property is a fluent one (and equivalently static objects when the property is not a fluent property), the language is also capable of handling the case where the property is a fluent one but is also connected with static objects in the ontology. In this case the query will retrieve both the dynamic and static objects it addresses. Summarizing, SOWL QL translation involves the following steps:

- Retrieves all dynamic objects (TimeSlices or Events) associated with a class in the static part of the ontology.
- Determines whether a property is fluent property or not.
- Works on the dynamic part of the ontology to answer to a query in cases where the property addressed in a fluent one.
- Works on the static part of the ontology to answer to a query, in cases where the property addressed is not a fluent (dynamic) one.
- Uses both the dynamic and static parts of the ontology to retrieve results in cases where the property addressed by the query is a fluent but is also connected with static objects.

In the example of Figure 5.3 the Company and Employee concepts are connected with dynamic objects (i.e., Timeslices in the case of the 4D-fluents or Events in the case of the N-ary model representation respectively). In order to retrieve the Employees that work for Company1, the query will retrieve both dynamic and static objects of class Employee (i.e., Employees working for Company1 even in cases where class Employee and the hasEmployee relation are defined as static).

```
SELECT ?employee
WHERE{
  ex:Company1 ex:hasEmployee ?employee
      }
```

Figure 5.3: Dynamic predicate triple example

Pattern 2:
Subject Predicate Object TemporalOperator(timepoint).

Operators in SOWL QL are distinguished into quantitative and qualitative. Quantitative operators have one or more timepoints as arguments. These can be further distinguished into timepoint operators if they have one argument(time instant) and, into time interval operators if the have two time points as arguments. When a temporal operator is used, the predicate of the triple must always be dynamic otherwise no results will be returned.

```
PATTERN 2:

Subject dynamicPredicate Object TemporalOperator(timepoint)
```

Figure 5.4: Triple pattern with timepoint quantitative temporal operator

The Timepoint operators are used to compare the intervals where the fluent property holds true with the timepoint that is specified as argument. Figure 5.4 illustrates the syntax of such queries. There are five cases:

- The timepoint argument equals starting point of the time interval that the fluent holds.
- The timepoint argument equals the the ending point of the time interval that the fluent holds.
- The timepoint argument is somewhere during the interval that the fluent holds.
- The timepoint argument is before the interval that the fluent holds.
- The timepoint argument is after the interval that the fluent holds.

Accordingly, SOWL QL implements the following five operators on time points: AT, STARTSAT, ENDSAT, AFTER, BEFORE. Figure 5.5 illustrates an example query that retrieves companies with employees before time point "2010-02-08T00:00:00" whose name is "Smith".

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee BEFORE("2010-02-08T00:00:00").
  ?employee ex:employeeName "Smith"
      }
```

Figure 5.5: Example of query with time point operator

Pattern 3:
Subject Predicate Object TemporalOperator(timepoint,timepoint).

Time interval operators take as arguments two time points, denoting the starting and the ending of a temporal interval. The interval composed by the two arguments imposes a restriction over temporal intervals where the dynamic predicate specified in the triplet holds true. SOWL QL implements the following quantitative operators over temporal intervals: ALWAYS_AT, SOMETIME_AT and all Allen operators. Each operator is a different restriction that is applied to the fluent property of the triple. For example the Allen operator EQUALS can be true only if it matches both the interval beginning and ending time instants of the temporal interval specified in the argument. Figure 5.6 illustrates the generic syntax of such query patterns.

```
PATTERN 3:

Subject dynamicPredicate Object TemporalOperator(timepoint1,timepoint2)
```

Figure 5.6: Triple specifying a time interval quantitative temporal operator on a temporal interval.

Figure 5.7 illustrates an example query that retrieves companies with employees during the specified temporal interval whose name is "Smith".

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee
  SOMETIME_AT("2010-02-08T00:00:00","2012-02-08T00:00:00")
  ?employee ex:EmployeeName "Smith"
    }
```

Figure 5.7: Example query with operator on time on temporal interval.

Pattern 4:
Subject1 Predicate1 Object1 TemporalOperator Subject2 Predicate2 Object2.

Qualitative temporal operators are used to constraint the relation between two triples. Each such triple represents a relation that holds over a specific time interval. All Allen operators can be used as qualitative operators in SOWL QL. Figure 5.8 illustrates the generic syntax pattern of qualitative temporal operators.

PATTERN 4:

```
Subject1 dynamicPredicate1 Object1
TemporalQualitativeOperator
Subject2 dynamicPredicate2 Object2
```

Figure 5.8: Qualitative temporal operator between two triples

Figure 5.9 illustrates an example query that retrieves companies with employees whose name is "Smith" and who had been working for these companies before they started to work for "Company1".

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee
  BEFORE
  ex:Company2 ex:hasEmployee ?employee2.
  ?employee2 ex:employeeName "Smith"
      }
```

Figure 5.9: Example query with qualitiative temporal operator.

Pattern 5:
Subject SpatialOperator Object.

The SOWL spatial representation introduced the "Location" and "Geometry" classes (see Sec. 3.1.3). Static spatial objects are connected with Location objects directly while dynamic spatial objects (moving objects) are connected with dynamic temporal objects (TimeSlices or Events) which are in turn connected to Location objects. Each Location has a geometry meaning that it is connected to a Geometry object using the "hasGeometry" property. Spatial relations are properties that connect Geometry objects.

Figure 5.10 illustrates a generic spatial query pattern involving a spatial operator (as a predicate) imposing a constraint on the relation between the subject and the object of a triple. This requires that the subject and the object are both connected to Location objects. If the subject or the object in the triple are not connected to Location objects it means that no spatial operator can be applied between them.

PATTERN 5:

```
Subject spatialPredicate Object
```

Figure 5.10: Triple with spatial operator

The spatial query of Figure 5.11 will retrieve countries in the north (Nof) of Greece.

```
SELECT ?country
WHERE {
  ?country spatial:Nof ex:Greece
        }
```

Figure 5.11: Example of spatial query.

Example of Figure 5.11 illustrates a static spatial operator (as countries aren't theoretically supposed to change location over time). However, spatial properties can also be dynamic just like all other properties. In this case, being dynamic means that, instead of connecting two Location objects, the spatial property connects two dynamic objects (TimeSlices or Events) which are in turn connected to Location objects. The query language is able to retrieve all those dynamic objects and combine the results with the static results. In such a case, the triples are considered to be (dynamic) spatio-temporal triples.

Pattern 6:
Subject SpatialOperator Point(x,y).

In this query pattern one spatial object of the triple is replaced by POINT(x,y). Here, instead of comparing the Geometry objects of two spatial objects we compare the geometry of the first spatial object with the Geometry of the Point specified. Figure 5.12 illustrates the generic syntax of spatial query with a quantitative spatial operator. The POINT(x,y) operator uses two arguments corresponding to the x and y axes in the two-dimensional space. Both arguments are float numbers. If the Point specified does not exist in the knowledge base, it is asserted by the query language and the reasoner is invoked to produce its relations with the existing objects. This is explained in detail in Section 5.4. The query of Figure 5.13 will retrieve countries north of point x,y. If the point is not in the knowledge base, prior to answering the query the reasoner has to be invoked to infer its relations with countries which are instances of the ontology.

```
PATTERN 6:

Subject spatialPredicate  POINT(x,y)
```

Figure 5.12: Triple with spatial quantitative operator

```
SELECT ?country
WHERE {
  ?country spatial:Nof POINT(332.2,122.4)
        }
```

Figure 5.13: Spatial query with quantitative spatial operator.

Pattern 7:
Subject spatialOperator Object temporalOperator.

Figure 5.14 illustrates a generic spatio-temporal query pattern. In all spatio-temporal triples the spatial dynamic objects are first retrieved and then the temporal operator's restriction is applied to the time intervals they hold true. The temporal operator in this pattern can be either a timepoint operator or a time interval. In both cases the spatial dynamic objects (Timeslices or Events of Geometry objects) are first retrieved and then the restriction is applied. In the example in figure 5.14 we seek to find which cars are north of "street1" at a specific timepoint.

```
PATTERN 7:

Subject spatialPredicate Object temporalOperator
```

Figure 5.14: Spatio-temporal pattern specifying the location of an object and also temporal operator restricting the time this location location holds true.

```
SELECT ?car
WHERE {
  ?car spatial:Nof ex:Street1
  AT("2010-02-08T00:00:00")
        }
```

Figure 5.15: Spatio-temporal triple using the AT temporal operator for restricting the location of an object.

Pattern 8:
Subject spatialOperator Point(x,y) temporalOperator.

Figure 5.16 illustrates a pattern which is similar to pattern 7 with the difference that a point is specified in the place of an instance. The example in figure 5.16 shows which cars are north a specific point in 2d-space at a specific timepoint.

PATTERN 8:

```
Subject spatialPredicate  POINT(x,y) temporalOperator
```

Figure 5.16: Spatio-temporal pattern specifying the location of an object by a point and also temporal operator restricting the time this location location holds true.

```
SELECT ?car
WHERE {
  ?car spatial:Nof POINT(12.5,22.4)
  AT("2010-02-08T00:00:00")
       }
```

Figure 5.17: Spatio-temporal triple example

Pattern 9:
Subject spatialOperator Object temporalOperator.

In the pattern of Figure 5.18 a temporal qualitative operator (Allen) is used to connect two spatial triples. First, the dynamic spatial objects from both triples will be retrieved. Then, the Allen operator will be applied on these to produce the final results. In the example of Figure 5.18 we seek to find which cars are north of "street1" before "person1" is south of "street2".

PATTERN 9:

```
Subject1 spatialPredicate1 Object1
ALLEN
Subject2 spatialPredicate2 Object2
```

Figure 5.18: Spatio-temporal triple with qualitative Allen temporal operator restricting the locations of two objects.

```
SELECT ?car
WHERE {
  ?car spatial:Nof ex:Street1 BEFORE ex:Person1 spatial:Sof ex:Street2
}
```

Figure 5.19: Spatio-temporal triple restricting the locations of two spatial objects using a temporal operator.

5.2 Temporal operators

Temporal operators in SOWL QL are distinguished into quantitative and qualitative. Quantitative temporal operators are further distinguished by the number of arguments they use (i.e., addressing time points or intervals respectively).

Time point quantitative operators:

- AT(timepoint): The fluent holds true during a time interval which contains the timepoint.
- STARTSAT(timepoint): The fluent holds true during a time interval which starts at timepoint.
- ENDSAT(timepoint): The fluent holds true during a time interval which ends at timepoint
- BEFORE(timepoint): The fluent holds true during a time interval which ends before timepoint.
- AFTER(timepoint): The fluent holds true in a time interval which starts after timepoint

In the example of figure 5.20 the query retrieves the employees that work for Company1 during any time interval starting at the time specified.

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  STARTS("2010-02-08T00:00:00")
      }
```

Figure 5.20: Query retrieving employees who started their work for Company1 at a specific time.

Time interval quantitative operators:

- ALWAYS_AT(intervalStarts,intervalEnds): Returns true for fluents holding true during intervals which contain all points of the interval in question ALWAYS_AT is defined as a combination of the following 4 Allen operators: CONTAINS, EQUALS, STARTEDBY, ENDEDBY. If any of these is true then ALWAYS_AT is also true.
- SOMETIME_AT(intervalStarts,intervalEnds): This is a combination of 9 Allen operators and returns true for fluents holding for intervals that share common time points with the interval in question. EQUALS, OVERLAPS, OVERLAPPEDBY, STARTS, STARTEDBY, ENDS, ENDEDBY, CONTAINS, DURING. If any of these is true then SOMETIMES_AT is true.
- MEETS(intervalStarts,intervalEnds): Returns true if the first time interval meets the second one.
- METBY(intervalStarts,intervalEnds): Returns true if the second time interval meets the first one.
- OVERLAPS(intervalStarts,intervalEnds): Returns true if the first time interval overlaps with the second one.
- OVERLAPEDBY(intervalStarts,intervalEnds): Returns true if the second time interval overlaps with the first one.
- DURING(intervalStarts,intervalEnds): Returns true if the first time interval is during the second one.
- CONTAINS(intervalStarts,intervalEnds): Returns true if the first time interval is contains the second one.

- STARTS(intervalStarts,intervalEnds): Returns true if the two time intervals start together.
- STARTEDBY(intervalStarts,intervalEnds): Returns true if the two time intervals start together.
- ENDS(intervalStarts,intervalEnds): Returns true if the two time intervals end together.
- ENDEDBY(intervalStarts,intervalEnds): Returns true if the two time intervals end together.
- EQUALS(intervalStarts,intervalEnds): Returns true if the first time interval equals the second one.

In the following example of Figure 5.21 the query returns the employees that work for Company1 for intervals that equal the one defined by the quantitative arguments.

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  EQUALS("2010-02-08T00:00:00","2012-02-08T00:00:00")
      }
```

Figure 5.21: Company has employee for a specific interval

Variables as timepoint arguments: In both cases of quantitative operators, timepoints can be replaced with variables in order to retrieve the time instants or intervals where the predicate (fluent property) holds true. As shown in Figure 5.22 the variables specified as arguments of the "SOMETIME_AT" operator take values to find out the starting and ending points of the time intervals during which the employee "Johnson" was working for company "C1".

```
SELECT ?x ?y
WHERE {
        ?company ex:hasEmployee ?employee SOMETIME_AT(?x,?y).
        ?company ex:companyName "C1"
        ?employee ex:employeeName "Johnson"
      }
```

Figure 5.22: Find the intervals that Employee "Johnson" works for Company "C1"

Qualitative operators: All Allen operators can be also used as qualitative operators. Qualitative operators are placed between two triples as stated in the query patterns description previously. In this case the interval restricting the duration of the predicate of the first triple is defined by the interval that the predicate of the second triple holds true.

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  DURING("2010-02-08T00:00:00","2012-02-08T00:00:00")
      }
```

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  DURING
  ex:Company2 ex:hasEmployee ex:Employee2
      }
```

Figure 5.23: Quantitative and qualitative operator comparison

For example, the first query in Figure 5.23, the interval restricting the duration of the hasEmployee relation is restricted by the arguments of the quantitative operator. In the second query, the interval that the fluent holds will be compared to the interval that the fluent of the second triple holds (the interval that Company2 has Employee2).

5.3 Spatial Operators

SOWL QL defines two types of spatial operators referred to as Directional (specifying the directional relation of two objects in space) and Topological (specifying the relative location of two objects).

- Directional: (Assume that we have two spatial objects A and B that are connected with the following spatial operators)
  - Nof: A is North of B
  - Sof: A is South of B
  - Eof: A is East of B
  - Wof: A is West of B
  - NWof: A is North and West of B
  - NEof: A is North and East of B
  - SEof: A is South East of B
  - SWof: A is South West of B
  - SameX: A and B have the same 'x' coordinate (e.g A(2,3) , B(2,5))
  - SameY: A and B have the same 'y' coordinate (e.g A(2,3) , B(5,3))
  - SameXY: A and B have the same center (e.g A(2,3) , B(2,3))
- The following topological operators are defined between any two objects A and B:
  - Contains (NTTPi): Object A contains object B (not touching).
  - ContainsTouches (TTPi): Object A contains and touches object B.
  - Disjoint (DC): Objects A and B are not one inside another and not touching.
  - Touches (EC): Objects A and B touch each other from outside.
  - Equals(EQ): A and B are the same size and occupy the same location on space.
  - IntoTouches(TPP): Object A is contained and touched by object B.
  - Whithin(NTTP): Object A is contained by object B
  - Overlaps(PO): Objects overlap.

5.4 Reasoning in SOWL QL

SOWL QL queries are translated into equivalent SPARQL queries. All quantitative operators in a SOWL QL query, are translated using qualitative relations making the translation more homogeneous, as both qualitative and quantitative operators are treated as qualitative, avoiding numerical comparisons. When the ontology is loaded into memory, the reasoner call computes all qualitative relations from the quantitative ones by making value comparisons.

Moreover, when a quantitative operator is used, the language query mechanism checks if the quantitative values specified as arguments exist in the knowledge base. If a value does not exist, it is asserted into the knowledge base and the reasoner is invoked so that the (qualitative) relations of the new temporal value with existing ones are derived and used for answering the query. For example, the query of Figure 5.24 retrieves employees that worked for a company before the time point specified. If time point "2010-02-08T00:00:00" is not in the ontology, the reasoner with insert it (temporarily) into the knowledge base and compute its relations with existing fluents prior to answering the query.

```
SELECT ?x ?y
WHERE {
        ?x ex1:hasEmployee ?y BEFORE("2010-02-08T00:00:00")
      }
```

Figure 5.24: Temporal query using the reasoner.

The same thing applies for spatial quantitative operators. For example, the query of Figure 5.25, retrieves objects which are "north of POINT(3.4,10.2)". In this case, similarly to our previous example, the point object specified will be asserted into the ontology (if not already there) and the reasoner will compute all the qualitative relations between the new POINT and all geometric objects in the ontology.

```
SELECT ?x ?y
WHERE {
        ?x spatial:Nof ?y POINT(3.4 ,10.5)
      }
```

Figure 5.25: Spatial query using the reasoner.

Finally the reasoning mechanism allows for retrieving results that implicitly satisfy the conditions imposed by the query. For example, if company C_1 has Employee E_1 for the interval $interval_1$, that spans throughout year 2007, and Employee E_2 for an unknown $interval_2$ that *contains* $interval_1$, then by using the point based reasoning mechanism it can be inferred that both $interval_1$ and $interval_2$ contain the time point in the previous query. Thus the results: C_1,E_1 and C_1,E_2 will be returned.

5.5 Equivalence to SPARQL

Syntactically, SOWL QL expressions form a strict superset of SPARQL expressions meaning that every valid SPARQL query is also a valid query in SOWL-QL. In the following we show that every valid SOWL QL query can be translated into an equivalent SPARQL query over the supported representations.

As we mentioned in the previous section, SPARQL queries are based on graph matching where the query criteria are given in the form of *subject - predicate - object* RDF triples (called basic graph patterns) inside the "WHILE" clause of SPARQL. SOWL QL syntax, uses the same graph patterns with the addition of SOWL QL operators at the end of each triple. In the following we show that these operators are also translated to basic graph patterns and we will do this by enumerating all the possible query patterns in SOWL QL by showing their translation to SPARQL triples. First we will show all the possible temporal triples, then the spatial and finally the spatio-temporal triples.

Translation of dynamic triples: Figure 5.26 illustrates a simple dynamic triple where the predicate is a fluent. It's translation for both the 4D-fluents and the N-ary models is illustrated as well.

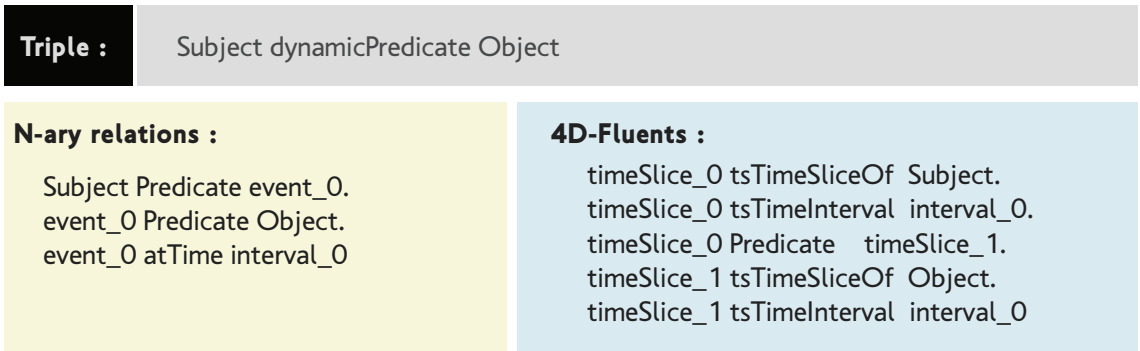

Figure 5.26: Dynamic triple translation

Translation of timepoint operators: Triples with timepoint operators are translated as shown in Figure 5.27. Notice that only part of the translation depends on the model while the rest (shown in green span in Figure 5.26) is exactly the same for both models. Figure 5.28 illustrates the translation for any temporal operator specifying a time point. According to the operator used, one of those translations will be used in the pattern of figure 5.27.

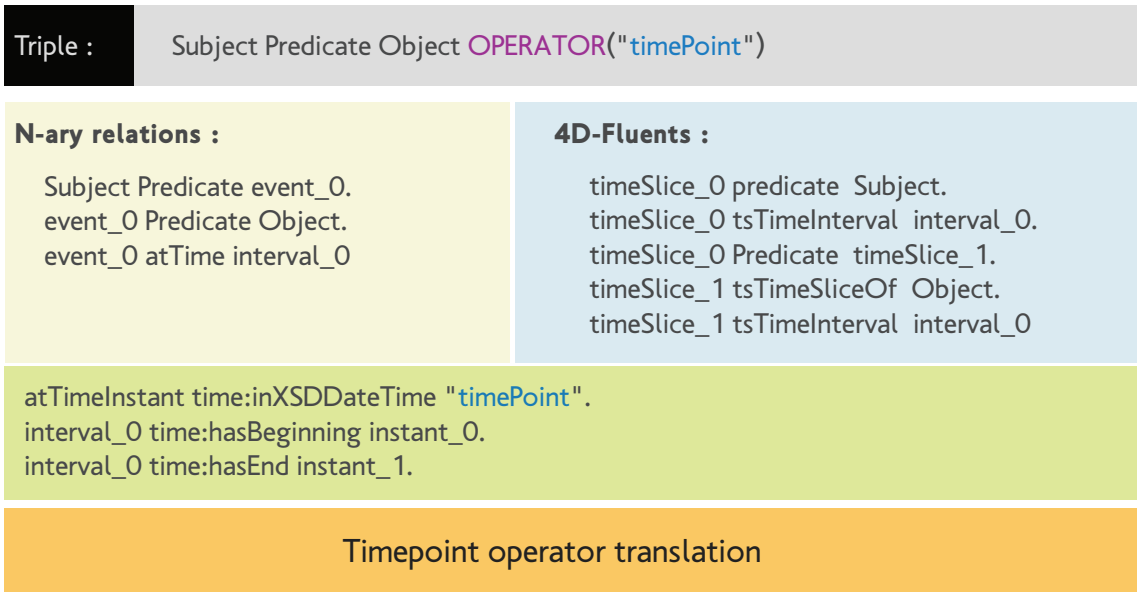

Figure 5.27: Temporal timepoint triple translation

TIMEPOINT OPERATORS TRANSLATION

AT :
```
{ instant_0 time:before atTimeInstant.
  instant_1 time:after atTimeInstant. }
UNION
{ instant_0 ex2:equals atTimeInstant.}
UNION
{?_instant_1 ex2:equals atTimeInstant.}
```

AFTER :
```
instant_0 time:after atTimeInstant.
```

BEFORE :
```
instant_1 time:before atTimeInstant.
```

STARTSAT :
```
instant_0 time:equals atTimeInstant.
```

ENDSAT :
```
instant_1 time:equals atTimeInstant.
```

Figure 5.28: Timepoint operators translation

Translation of time interval operators: Similar to the previous case, translating time intervals (fluents) to SPARQL triples is shown in figure 5.29 and the translation of each individual operator is shown in figures 5.30 and 5.31 (Allen).

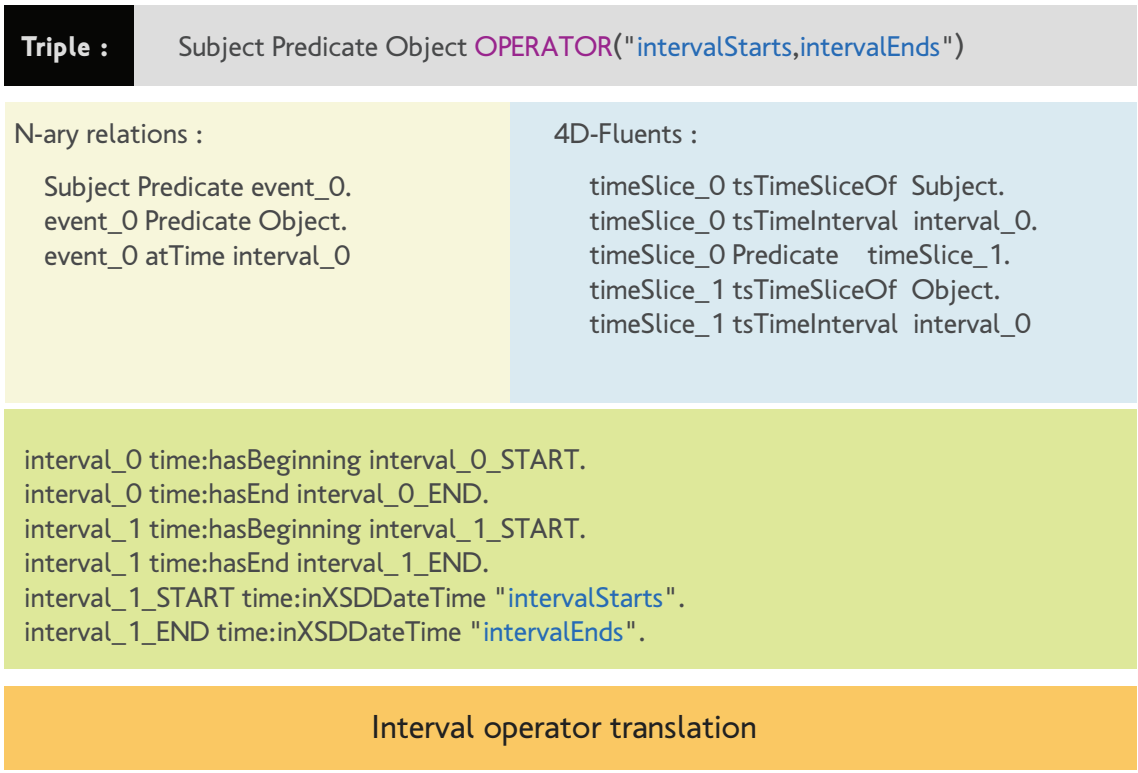

Figure 5.29: Temporal time interval triple translation

TIME INTERVAL OPERATORS TRANSLATION

SOMETIME_AT :

```
{ interval_0_START ex2:equals interval_1_START.
interval_0_END ex2:equals interval_1_END. }
UNION { interval_0_START time:before interval_1_START.
interval_0_END time:after interval_1_START.
interval_0_END time:before interval_1_END. }
UNION { interval_0_START time:after interval_1_START.
interval_0_START time:before interval_1_END.
interval_0_END time:after interval_1_END. }
UNION { interval_0_START ex2:equals interval_1_START.
interval_0_END time:before interval_1_END. }
UNION { interval_0_START ex2:equals interval_1_START.
interval_0_END time:after interval_1_END. }
UNION { interval_0_START time:after interval_1_START.
interval_0_END ex2:equals interval_1_END. }
UNION { interval_0_START time:before interval_1_START.
interval_0_END ex2:equals interval_1_END. }
UNION { interval_0_START time:before interval_1_START.
interval_0_END time:after interval_1_END. }
UNION { interval_0_START time:after interval_1_START.
interval_0_END time:before interval_1_END. }
```

ALWAYS_AT :

```
{ interval_0_START ex2:equals  interval_1_START.
 interval_0_END ex2:equals  interval_1_END. }
UNION { interval_0_START time:before  interval_1_START.
 interval_0_END time:after  interval_1_END. }
UNION {  interval_0_START ex2:equals  interval_1_START.
 interval_0_END time:after  interval_1_END. }
UNION {  interval_0_START time:before  interval_1_START.
 interval_0_END ex2:equals  interval_1_END. }
```

ALEN OPERATORS

Figure 5.30: Time interval operators translation

Allen operators translation	
BEFORE :	interval_0_END time:before interval_1_START.
AFTER :	interval_0_START time:after interval_1_END.
MEETS :	interval_0_END time:equals interval_1_START.
METBY :	interval_0_START time:equals interval_1_END.
OVERLAPS :	interval_0_START time:before interval_1_START. interval_0_END time:after interval_1_START. interval_0_END time:before interval_1_END.
OVERLAPPEDBY:	interval_0_START time:after interval_1_START. interval_0_START time:before interval_1_END. interval_0_END time:after interval_1_END.
DURING :	interval_0_START time:after interval_1_START. interval_0_END time:before interval_1_END.
CONTAINS :	interval_0_START time:before interval_1_START. interval_0_END time:after interval_1_END.
STARTS :	interval_0_START time:equals interval_1_START. interval_0_END time:before interval_1_END.
STARTEDBY :	interval_0_START time:equals interval_1_START. interval_0_END time:after interval_1_END.
ENDS :	interval_0_START time:before interval_1_START. interval_0_END time:equals interval_1_END.
ENDEDBY :	interval_0_START time:after interval_1_START. interval_0_END time:equals interval_1_END.
EQUALS :	interval_0_START time:equals interval_1_START. interval_0_END time:equals interval_1_END.

Figure 5.31: Allen operators translation

Translation of qualitative temporal operators: Qualitative operators are Allen operators and their translation was shown in the previous paragraph in Figure 5.31.

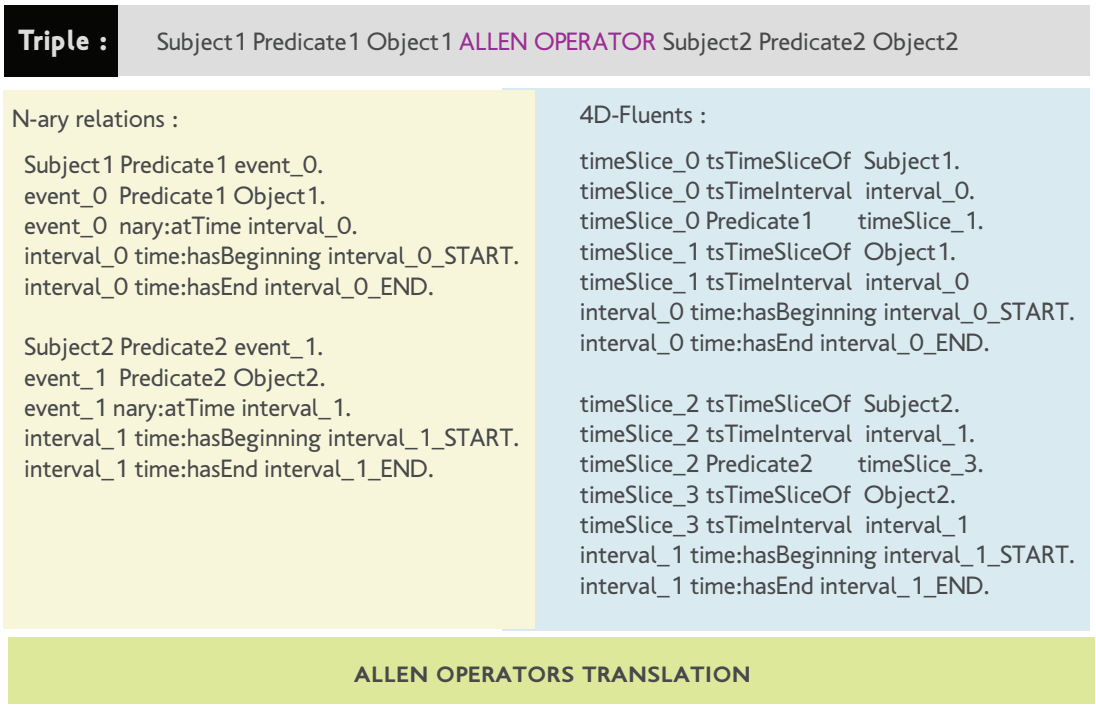

Figure 5.32: Qualitative temporal triple translation

Translation of spatial qualitative operators: The translation of spatial qualitative triples where the predicate is replaced by a spatial operator is shown in Figure 5.33. Notice that this is actually a spatio-temporal query because the translation along with the static results will retrieve dynamic (TimeSlices or Events) as well static objects satisfying the query selection criteria. Spatial operators are translated before temporal ones.

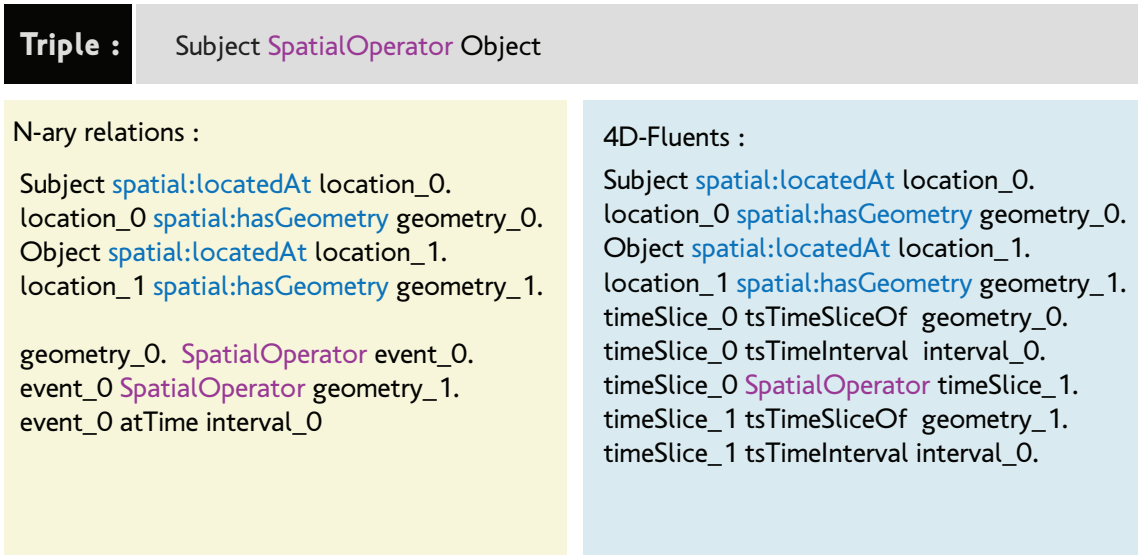

Figure 5.33: Qualitative spatial triple translation

Translation of spatial quantitative operators: Translation of a spatial triple using the Point(x,y) operator.

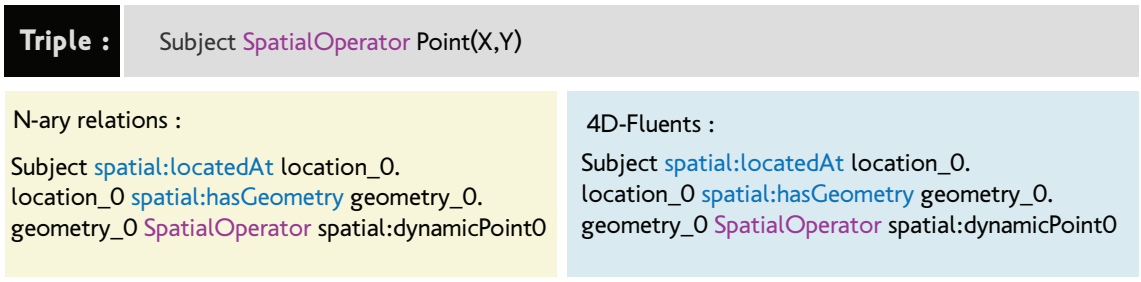

Figure 5.34: Quantitative spatial triple translation

Spatio-temporal queries translation Spatio-temporal queries are handled as spatial queries in the first place, and after the spatial translation is completed the temporal part of the query is translated as well. There is no specific pattern here as this type of translation is a combination of patterns shown above. Figure 5.35 illustrates an example query combining spatial and temporal query patterns.

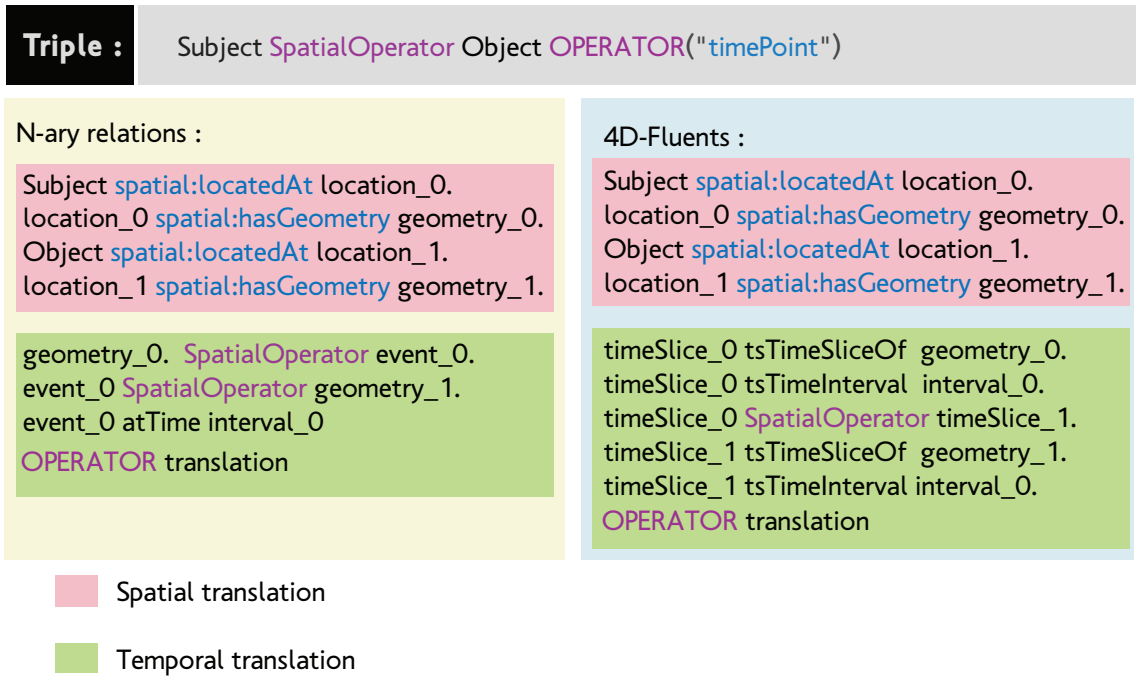

Figure 5.35: Spatio-temporal triple translation

Marked with red is the spatial translation (the triples needed to retrieve the spatial static objects) and with green the temporal translation corresponding to the models (4D-fluents or N-ary). The spatial part of the translation is exactly the same as it is independent from the temporal model and translation.

5.6 Examples

In the following examples we demonstrate most of SOWL QL's temporal and spatial operators. Notably we are going to illustrate some extra features like using variables instead of timepoint arguments or querying with timepoints or interval arguments that do not exist in the knowledge base (reasoner call). First we are going to give some temporal query examples, then some spatial and finally we are going to show how spatial and temporal operators can be combined to form spatio-temporal queries.

In the next four figures we show the data in our knowledge base. In Figure 5.36 we present the static objects, in Figures 5.37 and 5.38 we illustrate the temporal data (objects connected with TimeSlices) and in Figure 5.39 we show the spatial data. The model we are using for temporal representation is the 4D-fluents model

but the same example queries can also be executed over an ontology using the N-ary relations temporal model.

Data (Turtle Format) [Static Data **]**

```
@prefix  ex1:  <http://www.semanticweb.org/ontologies/2008/4/sowl-ontology.owl>

ex1:Company1 ex1:companyName "C1"
ex1:Company2 ex1:companyName "C2"
ex1:CompanyStatic ex1:companyName "C_STATIC"

ex1:Employee1 ex1:employeeName "John"
ex1:Employee2 ex1:employeeName "Mark"
ex1:Employee3 ex1:employeeName "John"
ex1:EmployeeStatic ex1:employeeName "Jack"

ex1:Product2 ex1:price  15.0
ex1:Product3 ex1:price  20.0
```

Figure 5.36: Data in Turtle format

Data (Turtle Format) [Temporal Data **]**

```
ex1:Company1TimeSlice1 ex1:tsTimeSliceOf ex1:Company1 .
ex1:Company1TimeSlice1 ex1:hasEmployee ex1:Employee1TimeSlice1 .
ex1:Company1TimeSlice1 ex1:tsTimeInterval ex1:Interval1 .

ex1:Company1TimeSlice2 ex1:tsTimeSliceOf ex1:Company1 .
ex1:Company1TimeSlice2 ex1:hasEmployee ex1:Employee2TimeSlice1 .
ex1:Company1TimeSlice2 ex1:tsTimeInterval ex1:Interval2 .

ex1:Company2TimeSlice1 ex1:tsTimeSliceOf ex1:Company2
ex1:Company2TimeSlice1 ex1:produces ex1:Product3TimeSlice1
ex1:Company2TimeSlice1 ex1:hasEmployee ex1:Employee3TimeSlice1 .
ex1:Company2TimeSlice1 ex1:tsTimeInterval ex1:Interval3 .

ex1:CompanyStatic ex1:hasEmployee ex1:EmployeeStatic .

ex1:Employee1TimeSlice1 ex1:tsTimeSliceOf ex1:Employee1 .
ex1:Employee1TimeSlice1 ex1:tsTimeInterval ex1:Interval1 .

ex1:Employee2TimeSlice1 ex1:tsTimeSliceOf ex1:Employee2 .
ex1:Employee2TimeSlice1 ex1:tsTimeInterval ex1:Interval2 .

ex1:Employee3TimeSlice1 ex1:tsTimeSliceOf ex1:Employee3 .
ex1:Employee3TimeSlice1 ex1:tsTimeInterval ex1:Interval3 .

ex1:Product1TimeSlice1 ex:tsTimeSliceOf ex:Product1
ex1:Product1TimeSlice1 ex:tsTimeIntervalOf ex:Interval1
ex1:Product1TimeSlice1 ex:productName “P1”
ex1:Product1TimeSlice1 ex:price 11.0

ex1:Product2TimeSlice1 ex:tsTimeSliceOf ex:Product2
ex1:Product2TimeSlice1 ex:tsTimeIntervalOf ex:Interval2
ex1:Product2TimeSlice1 ex:productName “P2”
ex1:Product2TimeSlice1 ex:price 21.0

ex1:Product3TimeSlice1 ex:tsTimeSliceOf ex:Product3
ex1:Product3TimeSlice1 ex:tsTimeIntervalOf ex:Interval3
ex1:Product3TimeSlice1 ex:productName “P3”
ex1:Product3TimeSlice1 ex:price 31.0

ex1:Product3TimeSlice2 ex:tsTimeSliceOf ex:Product3
ex1:Product3TimeSlice2 ex:tsTimeIntervalOf ex:Interval4
ex1:Product3TimeSlice2 ex:productName “P3x”
ex1:Product3TimeSlice2 ex:price 32.0
```

Figure 5.37: Data in Turtle format

Data (Turtle Format) [Temporal Data **] (cont)**

```
ex1:Instant1-2-7 xsd:inXSDDateTime “2007-02-01T00:00:00Z”.
ex1:Instant10-2-7 xsd:inXSDDateTime “2007-02-10T00:00:00”.
ex1:Instant13-2-07 xsd:inXSDDateTime “2007-02-13T00:00:00”.
ex1:Instant3-2-7 xsd:inXSDDateTime “2007-02-03T00:00:00”.
ex1:Instant5-2-7 xsd:inXSDDateTime “2007-02-05T00:00:00”.
ex1:Instant6-2-7 xsd:inXSDDateTime “2007-02-06T00:00:00”.
ex1:Instant7-2-7 xsd:inXSDDateTime “2007-02-07T00:00:00”.
ex1:Instant8-2-7 xsd:inXSDDateTime “2007-02-08T00:00:00”.

ex1:Interval1 ex1:hasBeginning ex1:Instant1-2-7
ex1:Interval1 ex1:hasEnd ex1:Instant5-2-7

ex1:Interval2 ex1:hasBeginning ex1:Instant6-2-7
ex1:Interval2 ex1:hasEnd ex1:Instant10-2-7

ex1:Interval3 ex1:hasBeginning ex1:Instant3-2-7
ex1:Interval3 ex1:hasEnd ex1:Instant7-2-7

ex1:Interval4 ex1:hasBeginning ex1:Instant8-2-7
ex1:Interval4 ex1:hasEnd ex1:Instant13-2-7
```

Figure 5.38: Data in Turtle format

Data (Turtle Format) [Spatial Data **]**

```
ex1:spatialObject1 ex1:locatedAt ex1:location1 .
ex1:spatialObject2 ex1:locatedAt ex1:location2 .
ex1:spatialObject3 ex1:locatedAt ex1:location3 .
ex1:spatialObject4 ex1:locatedAt ex1:location4 .
ex1:spatialObject5 ex1:locatedAt ex1:location5 .
ex1:spatialObject6 ex1:locatedAt ex1:location6 .
ex1:spatialObject7 ex1:locatedAt ex1:location7 .

ex1:location1 ex1:hasGeometry ex1:Envelope1 .
ex1:location2 ex1:hasGeometry ex1:Envelope2 .
ex1:location3 ex1:hasGeometry ex1:Polygon3 .
ex1:location4 ex1:hasGeometry ex1:Polygon4 .
ex1:location5 ex1:hasGeometry ex1:Polygon5 .
ex1:location6 ex1:hasGeometry ex1:Polygon1 .
ex1:location7 ex1:hasGeometry ex1:Point5 .

ex1:envelope1 spatial:NEpoint ex1:Point2 .
ex1:envelope1 spatial:SWpoint ex1:Point1 .
ex1:envelope2 spatial:SWpoint ex1:Point2 .
ex1:envelope2 spatial:NEpoint ex1:Point5 .
ex1:envelope3 spatial:Nof ex1:Envelope1 .

ex1:Polygon1 spatial:Nof ex1:Polygon2 .
ex1:Polygon3 spatial:Nof ex1:Polygon1 .
ex1:Polygon4Slice ex:tsTimeSliceOf ex1:Polygon4 .
ex1:Polygon4Slice ex:tsTimeIntervalOf ex1:Interval1 .
ex1:Polygon4Slice spatial:Nof ex1:Polygon5Slice .
ex1:Polygon5Slice ex:tsTimeSliceOf ex1:Polygon5 .
ex1:Polygon5Slice ex:tsTimeIntervalOf ex1:Interval1 .

ex1:Point1 spatial:x  25.4  .
ex1:Point1 spatial:y  36.45  .
ex1:Point2 spatial:x  26.2  .
ex1:Point2 spatial:y  38.5  .
ex1:Point3 spatial:Nof  ex1:Point2  .
ex1:Point3 spatial:Eof  ex1:Point2  .
ex1:Point4 spatial:Eof  ex1:Point3  .
ex1:Point4 spatial:y  45.2  .
ex1:Point5 spatial:x  21.7  .
ex1:Point5 spatial:y  32.5  .
ex1:Point6 spatial:x  20.0  .
ex1:Point6 spatial:y  30.0  .
```

Figure 5.39: Data in Turtle format

Example 1: In this example we seek the employees of a Company with name "C1". The hasEmployee predicate here is a dynamic predicate so the dynamic objects will be retrieved (TimeSlices). Notice that the static result EmployeeStatic is not retrieved because he works for a different company.

Query 1

```
select distinct  ?y
where{
        ?x ex1:hasEmployee ?y.
        ?x ex1:companyName "C1"
}
```

Results :

Employee2
Employee1

Figure 5.40: Example 1

Example 2: Here we seek to find all products which have a higher price than 10.0 and their prices. This query is very similar to the first with only deference that here the dynamic predicate is a datatype property, not an object property. We can see that the static objects (products) are retrieved as well as the dynamic objects (Product TimeSlices).

Query 2

```
select distinct  ?x ?y
where{
        ?x ex1:price ?y.
        Filter( ?x > 10.0 )
}
```

Results :

Product2	15.0
Product3	22.0
Product3	20.0

Figure 5.41: Example 2

Example 3: In this example we demonstrate the use of the AT(timepoint) operator asking for the names of the companies that have employees at this specific timepoint and the also the employee Names. The "C_STATIC" company result is retrieved because its a static result supposed to hold for all timepoints.

Query 3

```
select distinct  ?x ?y
where {
        ?t ex1:hasEmployee ?z  AT("2007-02-05T00:00:00").
        ?t ex1:companyName ?x
        ?z ex1:employeeName ?y
  }
```

Results :

C2	John
C1	John
C_Static	Jack

Figure 5.42: Example 3

Example 4: Here we use a time interval operator in this query to get the employee names and the companies they work for an interval that contains all points of the interval used as argument. Static results again are retrieved.

Query 4

```
select distinct  ?x ?y
where {
        ?t ex1:hasEmployee ?z
        ALWAYS_AT("2007-02-01T00:00:00Z", "2007-02-05T00:00:00").
        ?t ex1:companyName ?x
        ?z ex1:employeeName ?y
 }
```

Results :

C1	John
C_Static	Jack

Figure 5.43: Example 4

Example 5: This example demonstrates the use of a qualitative Allen operator. Here the query asks for the price that "Product1" had before "Employee2" was employed by "Company1".

Query 5

```
select ?price
where {
        ex1:Product1 ex1:price  ?price
       BEFORE
        ex1:Company1 ex1:hasEmployee ex1:Employee2
  }
```

Results :

11.0

Figure 5.44: Example 5

Example 6: In this query we use an Allen quantitative operator with a timepoint as argument. What is interesting with this query is that the timepoint that is used as argument does not exist in the knowledge base. In most query languages this query would end up with no results but is SOWL QL the query language asserts the timepoint and then calls the reasoner.

Query 6

```
select distinct  ?x ?y
where {
        ?t ex1:hasEmployee ?z  BEFORE("2007-02-07T00:00:00").
        ?t ex1:companyName ?x
        ?z ex1:employeeName ?y
  }
```

Results :

C1	John
C_Static	Jack

Figure 5.45: Example 6

Example 7: In example 7 we demonstrate the use of variables instead of timepoint arguments in quantitative temporal operators. Here we ask for all the employees that are working for "Company1" and the corresponding time intervals.

Query 7

```
select distinct ?e ?t1 ?t2
where {
        ex1:Company1 ex1:hasEmployee ?e  ALWAYS_AT( ?t1 , ?t2 )
      }
```

Results :

Employee1	2007-02-01T00:00:00Z	2007-02-05T00:00:00
Employee2	2007-02-06T00:00:00	2007-02-10T00:00:00

Figure 5.46: Example 7

Example 8: This is a qualitative spatial query. Here we ask what is north of "spatialObject5".

Query 8

```
select distinct ?x
where {
        ?x spatial:Nof spatial:spatialObject5
      }
```

Results :

SpatialObject4

Figure 5.47: Example 8

Example 9: This is a quantitative spatio-temporal query.

Query 9

```
select distinct  ?x
where {
        ?x spatial:Nof POINT([9.2,10.3]) SOMETIME_AT
       }
```

Results :
SpatialObject4

Figure 5.48: Example 9

Example 10: This is a qualitative spatio-temporal query where the before Allen operator connects two dynamic spatial triples. Here the query is "what objects are south of SpatialObject3 before spatialObject5 touches anything".

Query 10

```
select distinct   ?x ?z
where{
     spatial:SpatialObject3 spatial:Nof  ?x  BEFORE  spatial:spatialObject5 spatial:EC ?z
}
```

Results :

spatialObject6	spatialObject5
spatialObject6	SpatialObject2
spatialObject6	SpatialObject4
spatialObject6	SpatialObject3
spatialObject6	SpatialObject1
spatialObject6	spatialObject6

Figure 5.49: Example 10

Chapter 6

Implementation

6.1 The SOWL QL system.

As a proof of concept, we implemented a Graphical User Interface (GUI) on top of the SOWL QL query translator whose purpose is to facilitate expressing and executing of queries SOWL QL[1] Figure 6.1 illustrates the architecture of the SOWL QL system. The system consists of a GUI and three main modules namely the ontology Loader, the query parser and the interpreter. Each module performs specific tasks and is connected with the GUI for input and output. More Specifically:

1. The ontology loader module is used to load an ontology into memory and detect the underlying temporal or spatial model employed. It also plays a role during the parsing and the translation phases. During parsing, the ontology loader is used to determine if a predicate is a fluent and during translation the interpreter uses the ontology loader to retrieve information about the existence of a timepoint(which could be the start or end of an interval) or a spatial point in the knowledge base in order to call the reasoner or not.

2. The query parser is used to detect SOWL QL spatio-temporal operators in queries and use the corresponding interpreter (temporal, spatial or both).

3. The interpreter's task is to translate a SOWL QL query to the equivalent query in SPARQL and return the translated query.

The user uses the GUI to choose an OWL ontology to load. Then, the ontology loader module loads the ontology into memory and determines if a dynamic (temporal or spatial) model exists. If a dynamic model is recognized then, the ontology loader enables the appropriate parser (temporal, spatial or both) and also activates

[1]The SOWL QL system can be downloaded from http://www.intelligence.tuc.gr

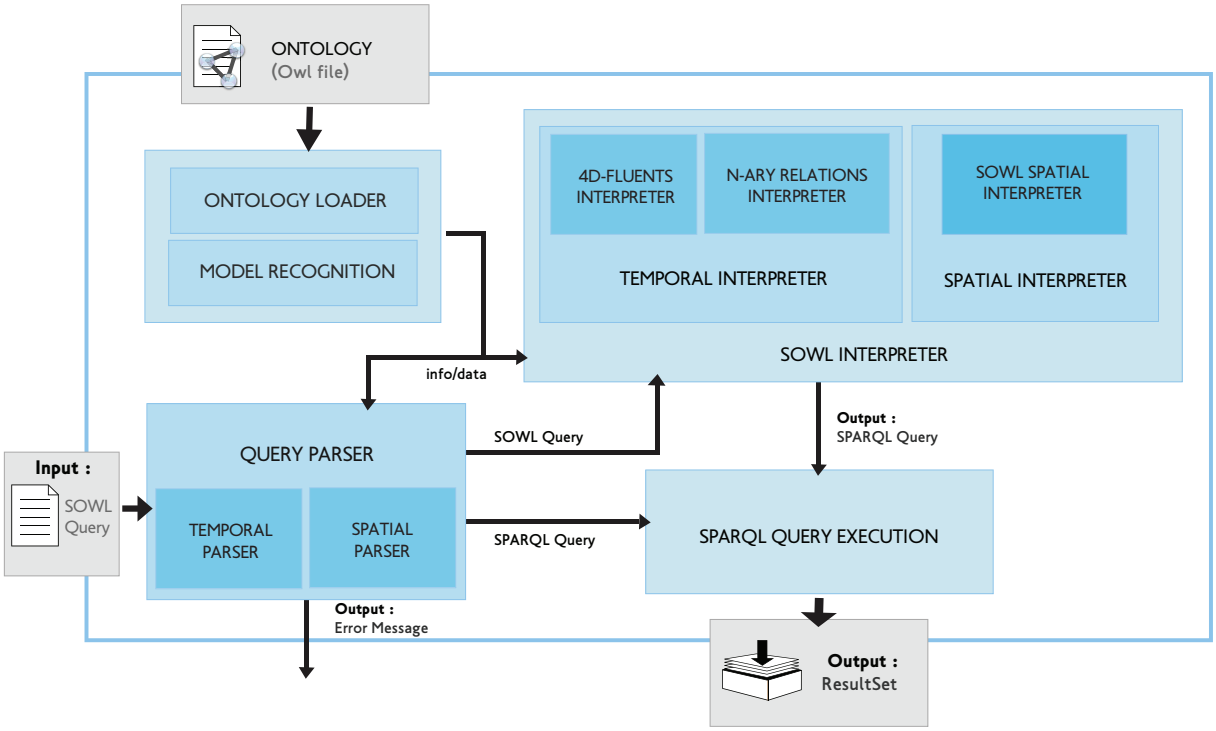

Figure 6.1: The SOWL system.

the corresponding temporal or spatial interpreter for the recognized models. For example, if the N-ary relations model is recognized then, the ontology loader will enable the temporal parser and the N-ary relations interpreter.

After the ontology is loaded into memory the user is able to use the GUI to write and execute queries. When a query is executed, the enabled parsers search for existing SOWL QL operators or dynamic predicates in the query. If no SOWL QL operators or dynamic predicates are found then, no translation takes place and the query is executed as a regular SPARQL query. If spatio-temporal operators or dynamic predicates are found then, the parser uses the appropriate interpreter to translate the query to SPARQL. When the translation process is completed, the translated SPARQL query is executed and the results are displayed by the GUI. In the following, role and of each module is discussed in more detail.

6.2 Ontology Loader

The ontology loader module executes 4 tasks:

1. Loads an OWL ontology into memory using the JENA API [75].

2. Identifies the underlying temporal or spatial model in the ontology, if one exists. This task is accomplished by parsing the concepts of the ontology and trying to identify one by one the URIs (Universal Resource Identifiers are unique strings that describes a resources) of the specific classes and properties that are required for each model representation. For example, in order to identify the 4D-fluents model, the "TimeSlice" and "Interval" classes and also the "tsTimeSlice" and "tsTimeInterval" properties should be identified. Figure 6.2 illustrates all the dynamic models that can be identified by the ontology loader module.

3. Informs the parser if the predicate in a subject - predicate - object triple is a fluent that is connected with dynamic object (TimeSlices or Events).

4. Informs the interpreter about the existence or not of timepoints or spatial points in the knowledge base.

TEMPORAL MODELS

4D-FLUENTS MODEL

CLASSES :

http://www.semanticweb.org/ontologies/2008/4/Ontology1211440295085.owl#TimeSlice
http://www.w3.org/2006/time#Interval

PROPERTIES :

http://www.semanticweb.org/ontologies/2008/4/Ontology1211440295085.owl#tsTimeSliceOf
http://www.semanticweb.org/ontologies/2008/4/Ontology1211440295085.owl#tsTimeInterval

N-ARY RELATIONS MODEL

CLASSES :

http://linkedevents.org/ontology/Event
http://www.w3.org/2006/time#Interval

PROPERTIES :

http://purl.org/NET/c4dm/event.owl#atTime

Figure 6.2: The URIs required to represent the 4D-fluents and the N-ary models.

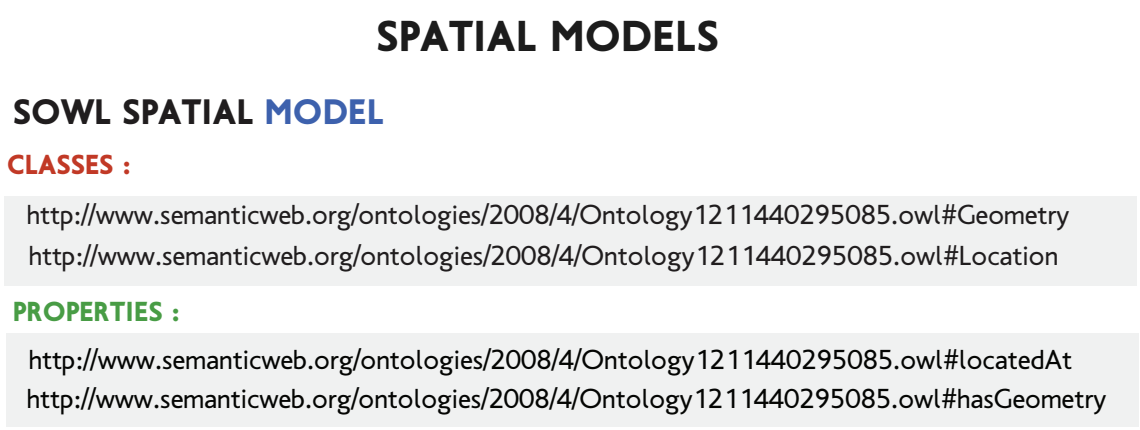

Figure 6.3: The URIs required to represent the SOWL spatial model.

The model identification process is used so the system can enable the appropriate parsers and interpreters in case a spatial or temporal model is identified. There are 4 cases:

1. No temporal or spatial model recognized: In this case no interpreters or parsers will be enabled. All the queries will be treated as SPARQL queries and will be executed as they are without any translation.

2. Temporal model recognized: In this case the temporal parser will be enabled and also the, corresponding to the recognized model, temporal interpreter (4D-fluents or N-ary relations interpreter).

3. Spatial model recognized: In this case only the spatial parser and the spatial interpreter will be enabled.

4. Spatio-temporal model recognized: In this case both the spatial and temporal parsers will be enabled and also the appropriate interpreters for both the spatial and temporal models that have been recognized.

Currently the system is able to identify the 4D-fluents and the N-ary temporal models and the SOWL spatial model.

6.3 Parser

The parser module is enabled only when a dynamic model is detected by the ontology loader. It consists of a temporal and a spatial parser which can be separately enabled or disabled. Each parser is capable of identifying a set of spatio-temporal

operators (presented in chapter 5) in the query and enable the appropriate interpreters according to the recognized models in order to translate it to SPARQL. Note that in spatio-temporal queries, where both the temporal and spatial parsers are enabled, the spatial parser always precedes the temporal parser.

As discussed in Chapter 5, a SOWL operator in a query can be used only after a triple pattern or between two triple patterns inside the *WHILE* clause. Moreover, the predicate of each triple can be a fluent that holds over a specific interval (e.g ex:Company ex:hasEmployee ex:Employee) or it can be a spatial operator (e.g ?x spatial:Nof ?y). The implementation of the parser is based on these simple acknowledgements. The parser scans the triples in the while clause sequentially and uses a look-ahead to read the next token after a triple. The following actions of the parser are determined by the look-ahead.

Temporal Parser: Subject Predicate object. In this case the look-ahead is not a SOWL QL temporal operator so the parser will check the predicate of the triple. If the predicate is dynamic (if it is a fluent that holds over a specific time interval) then the the corresponding temporal interpreter will be used to translate the triple into SPARQL. Otherwise, if the predicate is not dynamic, the parser moves to the next triple. To determine if the predicate is dynamic the parser uses data from the ontology loader to check if the predicate is connected with dynamic objects (timeslices in the 4D-fluents model or events in the N-ary relations model).

```
SELECT ?employee
WHERE{
  ex:Company1 ex:hasEmployee ?employee
      }
```

Figure 6.4: Dynamic predicate triple example

Here the *hasEmployee* predicate is dynamic and the query will return all the employees that had an employment relationship with *Company* during different time intervals.

Temporal Parser: Subject Predicate object TimePointOperator. In this case the operator has a single argument which is a timepoint. SOWL QL operators that can take timepoints as arguments are: AT, STARTSAT, ENDSAT, BEFORE and AFTER. The timepoint that is used as argument is in xsdDateTime format (e.g "2010-10-5T10:10:20Z").

```
SELECT ?employee
WHERE {
   ex:Company1 ex:hasEmployee ?employee AT("2010-02-08T00:00:00").
       }
```

Figure 6.5: Timepoint operator example

The above query will return all the employees that were working for *Company1* at the specified timepoint. If any Allen operators are used here as quantitative (instead of qualitative) the parser will use the look-ahead to check and confirm that quantitative timepoint arguments are used. If no quantitative arguments are found then the Allen operator is supposed to be qualitative and, in this case, it is handled differently. Operators AT, STARTSAT and ENDSAT are used only for quantitative time point temporal queries so no confirmation is needed.

Temporal Parser: Subject Predicate object TimeIntervalOperator. In this case the operator takes a time interval as argument consisting of two timepoints which are the starting and ending timepoints of an interval. SOWL QL operators that can take an interval as arguments are: ALWAYS_AT, SOMETIME_AT and all Allen operators. The timepoints that are used as arguments are again in xsdDateTime format.

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee
  SOMETIME_AT("2010-02-08T00:00:00","2012-02-08T00:00:00")
  ?employee ex:EmployeeName “Smith”
     }
```

Figure 6.6: Time interval operator example

Again here in case Allen operators are used the parser will use the look-ahead to determine if they are used as quantitative or qualitative.

Temporal Parser: Subject Predicate object Allen Subject2 Predicate2 Object2. This is the case where Allen operators are used as qualitative. In this

case the parser identifies an Allen temporal operator with no quantitative arguments. The look-ahead is used to assure that a second triple exists(3 more tokens) in order to call the interpreter.

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee
  BEFORE
  ex:Company2 ex:hasEmployee ?employee2.
  ?employee2 ex:employeeName "Smith"
      }
```

Figure 6.7: Qualitative temporal operator example

Spatial Parser: Subject Predicate object. In case the look-ahead is not a SOWL QL operator, the spatial parser will check if the predicate of the triple is a spatial operator (e.g spatial:Nof, spatial:Sof etc.) Then, if a spatial operator is recognized, the the corresponding spatial interpreter will be used to translate the triple to SPARQL. Otherwise, the parser moves to the next triple.

```
SELECT ?country
WHERE {
  ?country spatial:Nof ex:Greece
       }
```

Figure 6.8: Spatial operator example

Spatial Parser: Subject Predicate object PointSpatialOperator. In this case the look-ahead is a SOWL QL "Point(x,y)" operator with two arguments which are two float numbers corresponding to the x and y axis in 2D-space. An example of a spatial query of this type is illustrated in Figure 6.9

```
SELECT ?country
WHERE {
  ?country spatial:Nof POINT(332.2,122.4)
        }
```

Figure 6.9: Example spatial quantitative operator

Spatio-temporal queries: In spatio-temporal queries, the spatial parser always precedes the temporal. The spatial parser scans the query for spatial operators and calls the interpreters (as explained above). After spatial triple are translated, the temporal parser proceeds to scan the query and call the temporal interpreters to fully translate the query. Below we see two examples of spatio-temporal queries.

```
SELECT ?car
WHERE {
  ?car spatial:Nof ex:Street1
  AT("2010-02-08T00:00:00")
       }
```

Figure 6.10: Spatio-temporal triple example

```
SELECT ?car
WHERE {
  ?car spatial:Nof POINT(12.5,22.4)
  AT("2010-02-08T00:00:00")
       }
```

Figure 6.11: Spatio-temporal triple example

If no dynamic model is recognized then the parser module remains disabled and the query is treated as a standard SPARQL query. Also, there is an option through the GUI to disable all parsers and execute queries in SPARQL mode. This can be done by using the "SPARQL mode" button. When we choose this option, no matter if there is a dynamic model in the ontology, all queries are treated as

standard SPARQL queries and the use of SOWL QL operators will cause syntactic errors.

6.4 Interpreter

The interpreter module is the module that translates SOWL QL expressions into equivalent SPARQL queries. The implementation of the SOWL QL interpreter is illustrated in Figure 6.12. The main interpreter class is extended by two classes implementing the temporal and the spatial interpreters. The temporal and spatial interpreters are also extended by more specific interpreters according to the recognized dynamic model of the ontology as it is defined by the ontology loader module.

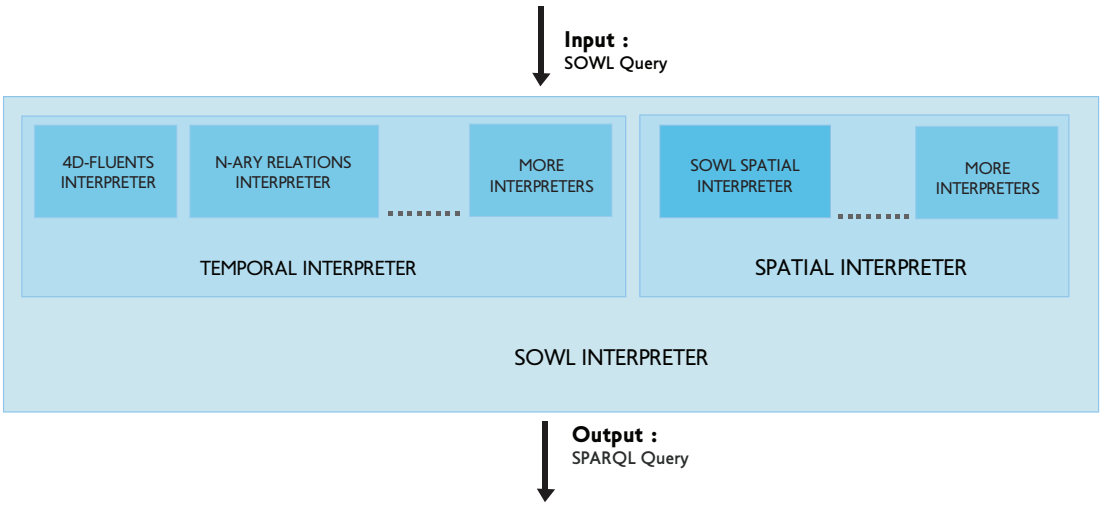

Figure 6.12: Sowl Interpreter.

The interpreters that are available to be used in the translation process are defined by the ontology loader module during the model recognition phase as we described previously. If for example, the ontology loader module detects the 4D-fluents model in the ontology then the 4D-fluents interpreter is enabled. All the methods that are used to translate SOWL QL queries are defined in the main SOWL Interpreter class and are the same for each specific interpreter implementation. As the methods are abstract, while they use the same arguments, their implementation varies according to the target model. Currently, the specific model interpreters that have been implemented are the 4D-fluents and N-ary relations temporal interpreters and and the SOWL QL spatial interpreter.

The interpreter is invoked by the parser whenever a triple with a fluent predicate or a SOWL QL operator is recognized. There are eight methods in total that the parser can call according to the query pattern that has been recognized. Six of those methods are used to translate temporal triples and two of them are used to translate spatial triples. Below we show the functions that are used to translate temporal and spatial triples.

translateDynamicTriple(String subject, String predicate, String object): This method is used to translate dynamic triples. An example translation of such a triple is illustrated in Figure 6.13

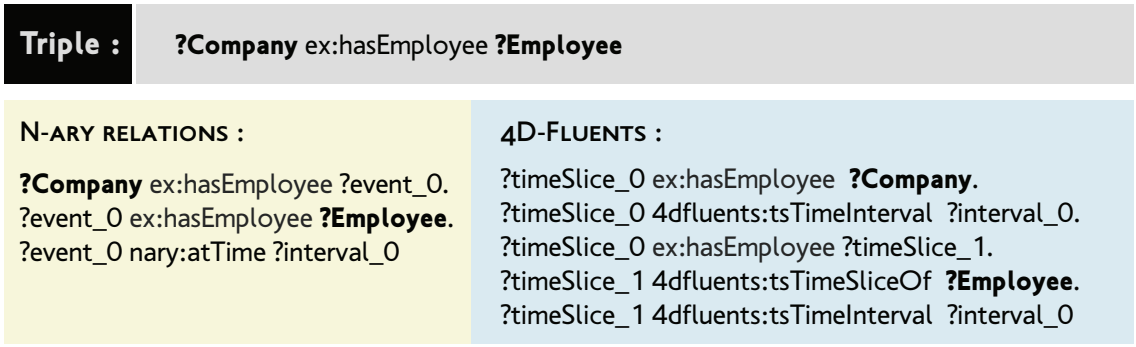

Figure 6.13: Dynamic triple translation example.

translateTimePointOperator(String subject, String predicate, String object, String timePoint): This method is used to translate a triple with a temporal timepoint operator. This method can handle the AT, BEFORE and AFTER temporal operators. We show two examples of timepoint triple translation, one in Figure 6.14 and one in Figure 6.15.

Triple : **?Company** ex:hasEmployee **?Employee** AT("2007-02-05T00:00:00")

N-ary relations :

?Company ex:hasEmployee ?event_0.
?event_0 ex:hasEmployee **?Employee**.
?event_0 nary:atTime ?interval_0

4D-Fluents :

?timeSlice_0 ex:hasEmployee **?Company**.
?timeSlice_0 4dfluents:tsTimeInterval ?interval_0.
?timeSlice_0 ex:hasEmployee ?timeSlice_1.
?timeSlice_1 4dfluents:tsTimeSliceOf **?Employee**.
?timeSlice_1 4dfluents:tsTimeInterval ?interval_0

```
?_atTimeInstant time:inXSDDateTime "2007-02-05T00:00:00"^^xsd:dateTime.
?_interval_0 time:hasBeginning ?_instant_1.
?_interval_0 time:hasEnd ?_instant_2.
{ ?_instant_1 time:before ?_atTimeInstant.
  ?_instant_2 time:after ?_atTimeInstant.}
UNION
{ ?_instant_1 ex2:equals ?_atTimeInstant.}
UNION
{?_instant_2 ex2:equals ?_atTimeInstant.}
```

Figure 6.14: Translation of a temporal triple using the AT(timepoint) temporal operator.

Triple : **?Company** ex:hasEmployee **?Employee** AFTER("2007-02-05T00:00:00")

N-ARY RELATIONS :

?Company ex:hasEmployee ?event_0.
?event_0 ex:hasEmployee **?Employee**.
?event_0 nary:atTime ?interval_0

4D-FLUENTS :

?timeSlice_0 ex:hasEmployee **?Company**.
?timeSlice_0 4dfluents:tsTimeInterval ?interval_0.
?timeSlice_0 ex:hasEmployee ?timeSlice_1.
?timeSlice_1 4dfluents:tsTimeSliceOf **?Employee**.
?timeSlice_1 4dfluents:tsTimeInterval ?interval_0

?_atTimeInstant time:inXSDDateTime "2007-02-05T00:00:00"^^xsd:dateTime.
?_interval_0 time:hasBeginning ?_instant_0.
?_interval_0 time:hasEnd ?_instant_1.
?_instant_0 time:after ?_atTimeInstant.

Figure 6.15: Translation of a temporal triple using the Allen operator AFTER.

translateDynamiTimeIntervalOperator(String subject, String predicate, String object, String intervalStarts, String intervalEnds): This method is used to translate triples concerning interval operators. Those are the ALWAYS_AT, SOMETIME_AT and all Allen operators which take two timepoints as arguments (the start and ending timepoints of an interval). An example translation is shown in Figure 6.16

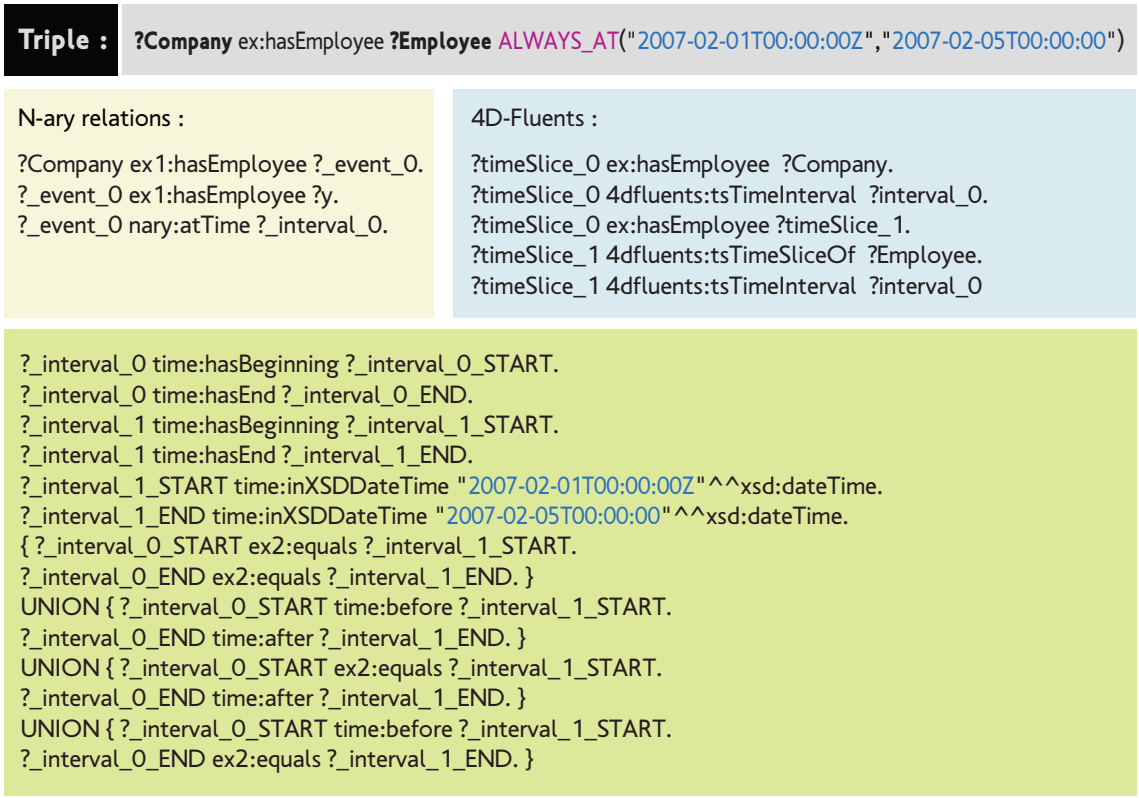

Figure 6.16: Translation of a temporal triple using the ALWAYS_AT temporal operator.

translateQualitativeOperator(String subject, String predicate, String object, String subject2, String predicate2, String object2, String allenOperator): This method is applied to translate qualitative temporal operators used between two triples. Figure 6.17 shows an example translation of such a triple.

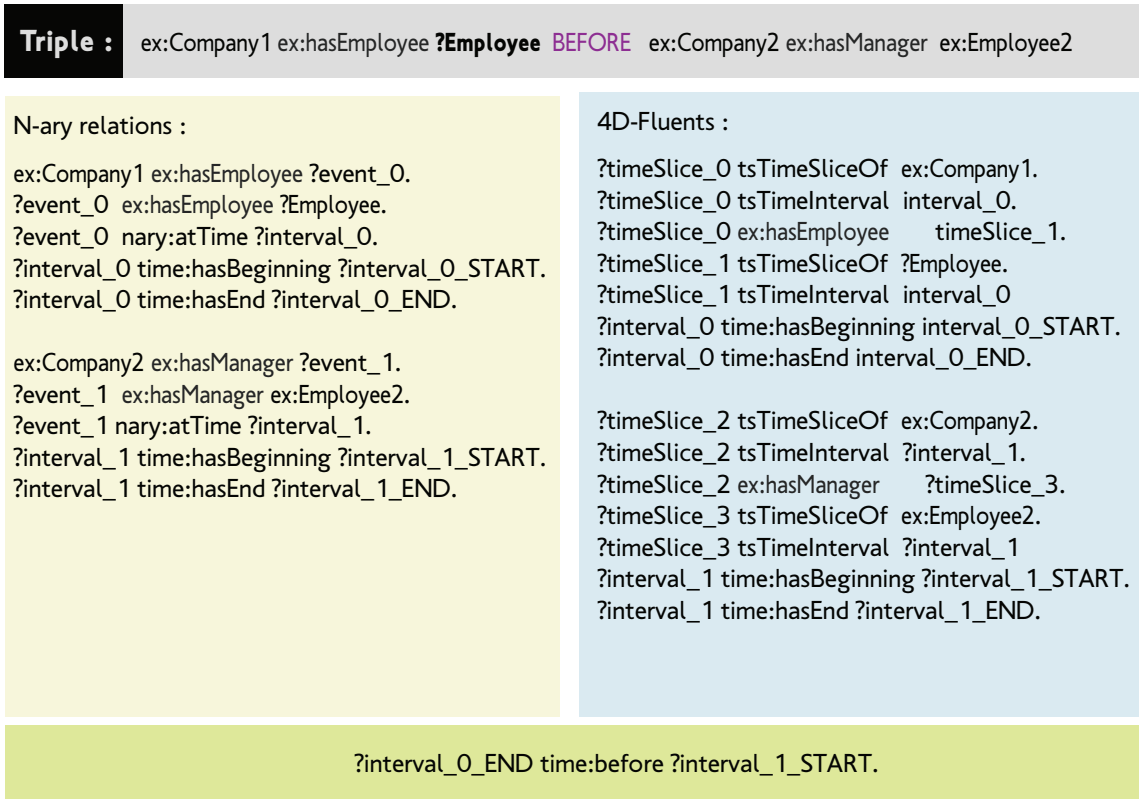

Figure 6.17: Translation of a qualitative temporal operator connecting two triples.

translateSpatialQualitative(String subject,String spatialOperator,String object): This method translates triples with a spatial operator as predicate (e.g ?x spatial:nof ?y). An example is shown in Figure 6.18.

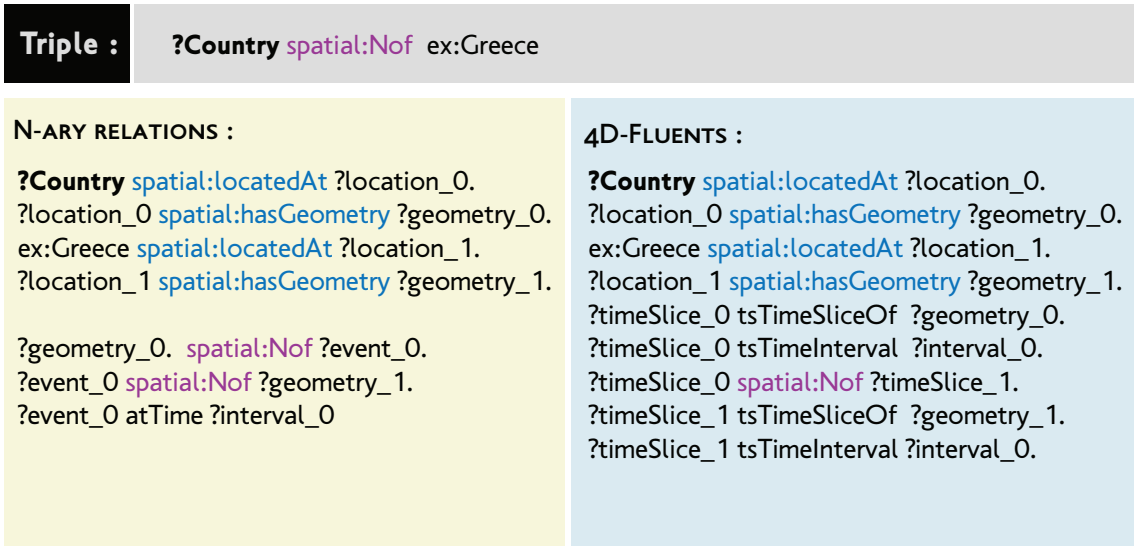

Figure 6.18: Translation of a qualitative spatial operator.

translateSpatialQuantitative(String subject,String predicate,String object,double pointX,double pointY): This method translates spatial quantitative triples. The POINT(x,y) operator in the place of the object in a triple (e.g ?x spatial:nof POINT(2.32,57.4)). An example is illustrated in Figure 6.19.

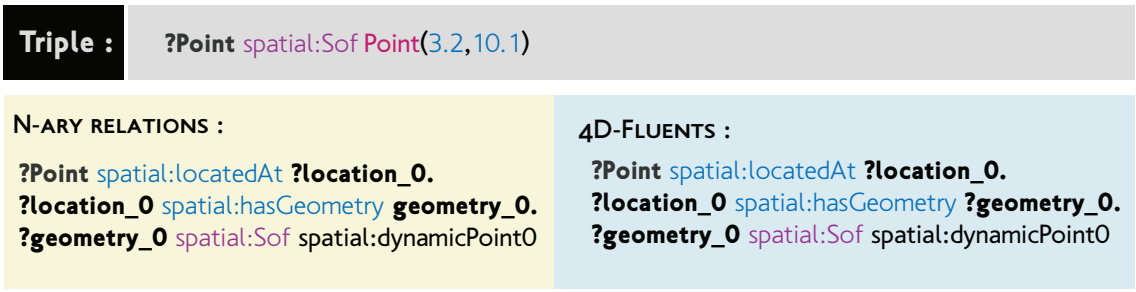

Figure 6.19: Translation of quantitative spatial operator.

6.5 GUI

The Graphical user interface provides the users with a more convenient way to place and execute SOWL QL queries. It accepts user input and combines all the other modules to produce the final output (the results of the query execution).

Figure 6.20 illustrates a screenshot of the GUI as it is when the JAVA application is executed. Figure 6.21 shows the GUI in full action (an ontology is loaded and a query is executed) and enumerates its main parts.

1. **Ontology Textfield:** This textfield displays the full path name of the ontology that is loaded. If the user tries to load an invalid file or an inconsistent ontology then the message "Ontology failed to load" is displayed at the textfield.

2. **Ontology loading flag:** A green light is displayed if the ontology is successfully loaded, otherwise a red light is displayed.

3. **Main menu buttons:** The main menu of the application. It consists of three buttons.

   - **Load Ontology:** This button opens a dialogue showing the system files so the user can browse and choose an OWL ontology file. By default it is programmed to open a dialogue showing the files under the "ontology files" directory which is inside the root folder of the application.
   - **Execute Query:** Executes the query that is written in the query text area.
   - **Exit Program:** Closes the application.

4. **Query Editing buttons:** A set of buttons to help with query editing.

   - **New Query:** Displays a dialogue asking the user to save the contents of the query text area and the clears the area so a new query can be typed.
   - **Load Query:** Opens a dialogue showing the files in the "query files" directory (which is also inside the root folder of the application) so the user can load a query. All SOWL QL queries have the .sowl extension.
   - **Save query:** Saves the current query in the text area.
   - **Save as query:** Prompts the user to save the query in the text area as a file in the "query files" folder.
   - **Text editing buttons:** Basic text manipulation. Cut, copy, paste, select all, undo, redo.

5. **Query text area:** This is the area where queries are written. Syntax highlighting has been implemented so the user can distinguish variables, datatypes, keywords etc.

6. **Results tab:** The results tab is the first of the two tabs of the interface's lower panel. This is where the results of the query execution are shown in a tabular form.

7. **Messages tab:** The second tab of the lower panel of the interface is used to display messages regarding the query execution. If the query is incorrect then error messages are displayed here. Otherwise, if the query is correct, the translated query in SPARQL is displayed.

8. **Interpreters panel:** In this panel the system displays information about the available interpreters. There are two smaller panels here. One for the temporal interpreter that is used and one for the spatial one. If a temporal or spatial model is recognized, by the ontology loader module, then the corresponding interpreter is enabled and the current temporal or spatial model (e.g 4D-fluents) is displayed. If no temporal or spatial model is displayer then both interpreters are shown as disabled and a third panel pops up informing the user that the system runs in "SPARQL mode", meaning that all queries will be treated as simple SPARQL queries because no dynamic models exist in the ontology.

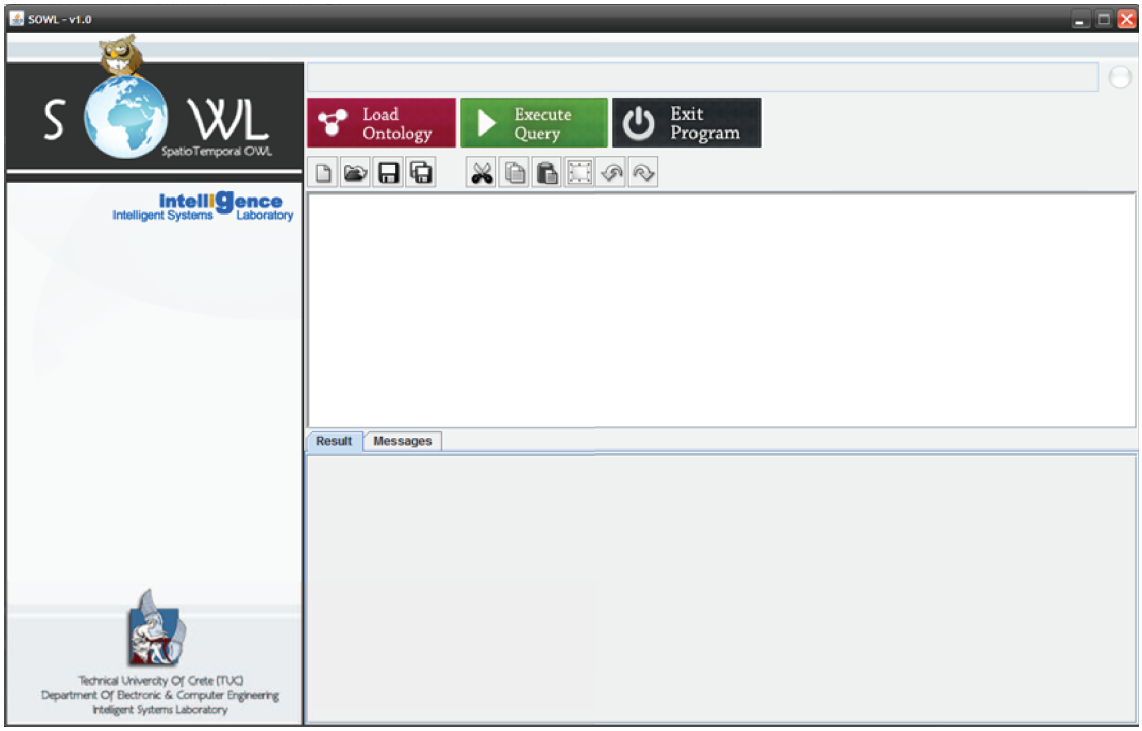

Figure 6.20: SOWL QL GUI: No ontology loaded.

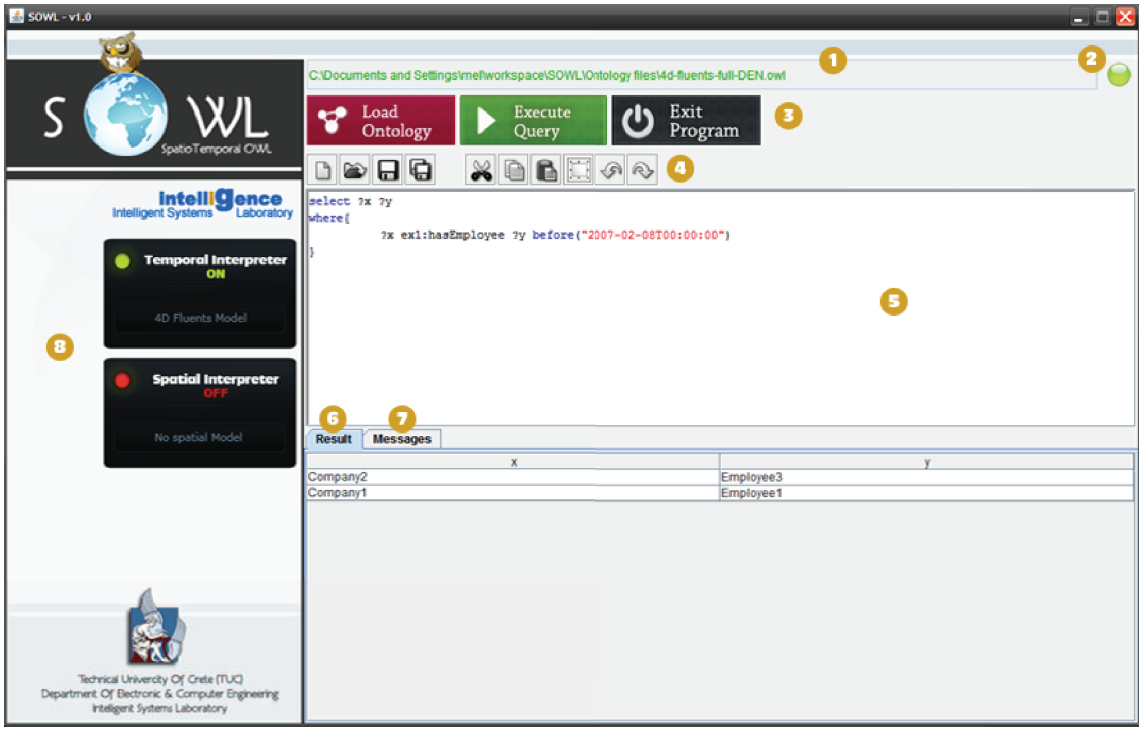

Figure 6.21: SOWL QL GUI: Query execution.

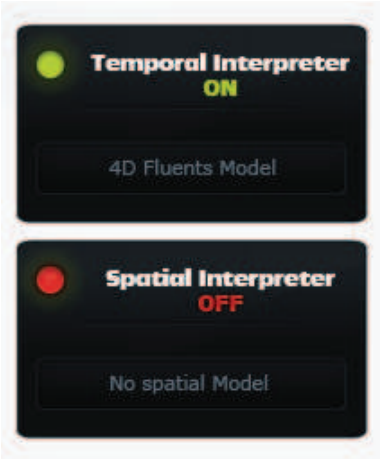

Figure 6.22: Interpreters Panel

The GUI is implemented using the JAVA Swing package.

Chapter 7

Conclusions and future work

In this work we introduced SOWL QL, a spatio-temporal query language which extends SPARQL by enabling spatio-temporal data querying. The proposed query language is fully compliant with the semantic web standards and extends SPARQL with a powerful set of spatiotemporal operators enabling it to query dynamic data. The query language is part of SOWL [10], an approach for handling temporal and spatial knowledge in ontologies. SOWL handles both, time instants and temporal intervals (and also semi-closed intervals) equally well using a sound and complete inference procedure based on path consistency and also handles property restrictions on fluent properties. SOWL applies on both, spatial and spatio-temporal information, by offering support for representing and reasoning over topological and directional spatial relations.

SOWL QL supports a powerful set of quantitative (interval and timepoint based) and qualitative temporal operators and also supports a set of topological and directional spatial operators. Also, SOWL QL offers reasoning support through the querying process.

Existing query languages such as t-SPARQL [109] and T-SPARQL [38], built-upon representations such as named graphs and versioning respectively, do not offer the expressive power of SOWL QL. Neither supports queries over qualitative information, nor they are supported by reasoning over a rich in spatio-temporal semantics ontological framework such as SOWL.

To show proof of concept, all language features are implemented in full and supported by a software system for SOWL QL translation and processing consisting of a query parser, an interpreter and a Graphical User Interface (GUI). The SOWL QL system is available on the Web[1]

[1] http://www.intelligence.tuc.gr

7.1 Conclusions

Summarizing, the contributions of this work are the following:

- SOWL QL is model independent so users don't need to be familiar with the underlying model representation. This is an advantage over languages like SPARQL where knowing the model representation is required to be able to write spatio-temporal queries.
- The language is capable of querying over both quantitative and qualitative spatial and temporal information. SOWL QL currently offers a range of operators wider than any other query language.
- SOWL QL supports quantitative temporal operators, all ALLEN operators (as qualitative and quantitative) as well as, topological and directional spatial operators.
- SOWL QL provides reasoning support to queries: If the quantitative argument of spatial or temporal operator is not explicitly defined, in the knowledge base, then, prior to answering the query, the argument is first asserted into the knowledge based and the reasoner is called for computing its relations with existing temporal or spatial information. Currently, all existing query languages cannot handle such queries unless quantitative data given as arguments are already in the knowledge base.
- SOWL is fully compliant with existing Semantic Web standards and specifications. Being compatible with W3C specifications SOWL can be used in conjunction with existing editors, reasoners and querying tools such as Protege and Pellet without requiring specific software.

7.2 Future work

The following are important issues for future work:

- Extending our current spatial operators to work with arbitrary geometries (e.g., polygons as stSPARQL [64] does) is an important issue for future work. Right now the language is only capable of using points in spatial quantitative queries.
- Query optimization is also an important issue for future work. Currently, when executing spatio-temporal queries, in SOWL QL, the spatial parser is always used before the temporal one during translation. A query optimizer

may check on the size of partial answers produced by each query pattern or, check on the number of spatial and temporal triples in a query and decide which parser is better to use first.

- An almost orthogonal issue (to the above) is speed of search. Query response time may be speeded-up significantly by incorporating into the query search process an indexing mechanism on point and interval-based spatial and temporal information. Currently when we are executing a temporal query all intervals stored in the knowledge base are processed sequentially (i.e., all intervals in the knowledge base are searched every time a new query is issued).

References

[1] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, **26**[11]:832–843, 1983.

[2] A. Ankolekar, M. Burstein, et al. DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference*, pages 348–363. Citeseer, 2002.

[3] A. Artale and E. Franconi. A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence*, **30**[1]:171–210, 2000.

[4] F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Pr., 2003.

[5] F. Baader. Description Logics. In *Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009*, **5689** of *Lecture Notes in Computer Science*, pages 1–39. Springer–Verlag, 2009.

[6] P. Balbiani, J.F. Condotta, and L.F. del Cerro. A Model for Reasoning about Bidimensional Temporal Relations. In *International Conference On Principles Of Knowledge Representation And Reasoning*, pages 124–130. Citeseer, 1998.

[7] P. Balbiani, J.F. Condotta, and L. Farinas del Cerro. A new Tractable Subclass of the Rectangle Algebra. In *International Joint Conference On Artificial Intelligence*, **16**, pages 442–447. Citeseer, 1999.

[8] E. Baratis, E. Petrakis, S. Batsakis, N. Maris, and N. Papadakis. Toql: Temporal Ontology Querying Language. *11th International Symposium on Spatial and Temporal Databases (SSTD 2009)*, pages 338–354, 2009.

[9] R. Batres, M. West, D. Leal, D. Price, K. Masaki, Y. Shimada, T. Fuchino, and Y. Naka. An Upper Ontology Based on ISO 15926. *Computers and Chemical Engineering*, **31**[5-6]:519 – 534, 2007.

[10] S. Batsakis. *SOWL: A Framework for Handling Spatio-Temporal Information in OWL*. PhD thesis, Technical Univercity Of Crete, 2011.

[11] S. Batsakis and E.G.M. Petrakis. Representing and Reasoning over Spatio-Temporal Information in OWL 2.0. *6th Workshop on Semantic Web Applications and Perspectives (SWAP' 2010)*, 2010.

[12] S. Batsakis and E.G.M. Petrakis. SOWL: Representing Spatio-Temporal Information in OWL. In *7th Extended Semantic Web Conference (ESWC'2010). Poster Paper*, 2010.

[13] S. Batsakis and E.G.M. Petrakis. SOWL: Spatio-Temporal Representation, Reasoning and Querying Over the Semantic Web. In *Proceedings of the 6th International Conference on Semantic Systems, (I-SEMANTICS' 2010)*, pages 15:1–15:9. ACM, 2010.

[14] S. Batsakis and E.G.M. Petrakis. Integrated Ontologies for Spatial Scene Descriptions. In *Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions, IGI Global (book chapter), S. M Hazarika (Eds)*. Citeseer, 2011.

[15] S. Batsakis and E.G.M. Petrakis. SOWL: A Framework for Handling Spatio-Temporal Information in OWL 2.0. *5th International Symposium on Rules: Research Based and Industry Focused (RuleML' 2011)*, pages 242–249, 2011.

[16] S. Batsakis, E.G.M. Petrakis, and E. Milios. Improving the Performance of Focused Web Crawlers. *Data & Knowledge Engineering*, **68**[10]:1001–1013, 2009.

[17] S. Batsakis, K. Stravoskoufos, and E.G.M. Petrakis. Temporal Reasoning for Supporting Temporal Queries in OWL 2.0. *15th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES' 2011)*, **6881**:558–567, 2011.

[18] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, et al. OWL Web Ontology Language Reference. *W3C recommendation*, **10**:2006–01, 2004.

[19] T. Berners-Lee, J. Hendler, O. Lassila, et al. The Semantic Web. *Scientific American*, **284**[5]:28–37, 2001.

[20] M. Bodirsky and H. Chen. Qualitative Temporal and Spatial Reasoning Revisited. *Journal of Logic and Computation*, **19**:1359–1383, December 2009.

[21] M.H. Bohlen and C.S. Jensen. Seamless Integration of Time Into SQL. In *Technical Report TR-962049, Aalborg University, Department of Computer Science*. Aalborg University, Institute for Electronic Systems, Dept. of Mathematics and Computer Science, 1996.

[22] J. Broekstra and A. Kampman. SeRQL: A Second Generation RDF Query Language. In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14. Citeseer, 2003.

[23] I. Budak Arpinar, A. Sheth, C. Ramakrishnan, E. Lynn Usery, M. Azami, and M.P. Kwan. Geospatial Ontology Development and Semantic Analytics. *Transactions in GIS*, **10**[4]:551–575, 2006.

[24] P. Buneman and E. Kostylev. Annotation Algebras for RDFS. In *The Second International Workshop on the Role of Semantic Web in Provenance Management (SWPM-10), CEUR Workshop Proceedings*, 2010.

[25] S. Chakrabarti, M. Van den Berg, and B. Dom. Focused Crawling: A New Approach to Topic-specific Web Resource Discovery. *Computer Networks*, **31**[11-16]:1623–1640, 1999.

[26] P.A. Champin and A. Passant. SIOC in Action Representing the Dynamics of Online Communities. In *Proceedings of the 6th International Conference on Semantic Systems*, pages 1–7. ACM, 2010.

[27] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. *Mobile and Ubiquitous Systems, Annual International Conference on*, pages 258–267, 2004.

[28] P.P. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, **1**[1]:9–36, 1976.

[29] S. Christodoulakis, M. Foukarakis, L. Ragia, H. Uchiyama, and T. Imai. Semantic Maps and Mobile Context Capturing for Picture Content Visualization and Management of Picture Databases. *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pages 130–136, 2008.

[30] G. Christodoulou. A Reasoning and Query Engine over Qualitative Spatial Information. *Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete (to appear)*, 2012.

[31] A.G. Cohn, B. Bennett, J. Gooday, and N.M. Gotts. Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. *GeoInformatica*, **1**[3]:275–316, 1997.

[32] A.G. Cohn and S.M. Hazarika. Qualitative Spatial Representation and Reasoning: An Overview. *Fundamenta Informaticae*, **46**[1-2]:1–29, 2001.

[33] W. Cowley and D. Plexousakis. Temporal Integrity Constraints with Indeterminacy. *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00*, pages 441–450, 2000.

[34] H.B. Enderton. *A Mathematical Introduction to Logic*, **40**. Academic press New York, 1972.

[35] D. Fensel, F. Van Harmelen, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *Intelligent Systems, IEEE*, **16**[2]:38–45, 2001.

[36] A.U. Frank. Qualitative Spatial Reasoning: Cardinal Directions as an Example. *International Journal of Geographical Information Science*, **10**[3]:269–290, 1996.

[37] F. Frasincar, V. Milea, and U. Kaymak. tOWL: Integrating Time in OWL. *Semantic Web Information Management: A Model-Based Perspective*, pages 225–246, 2010.

[38] F. Grandi. T-SPARQL: a TSQL2-like Temporal Query Language for RDF. In *International Workshop on on Querying Graph Structured Data*, pages 21–30, 2010.

[39] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The Next Step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, **6**[4]:309–322, 2008.

[40] H. Gregersen and C.S. Jensen. Temporal Entity-Relationship Models-a Survey. *Knowledge and Data Engineering, IEEE Transactions on*, **11**[3]:464–497, 1999.

[41] P. Grenona and B. Smith. SNAP and SPAN: Towards Dynamic Spatial Ontology. *Event-Oriented Approaches in Geographic Information Science: A Special Issue of Spatial Cognition and Computation*, **4**[1]:69–104, 2004.

[42] R. Grutter and B. Bauer-Messmer. Towards Spatial Reasoning in the Semantic Web: A Hybrid Knowledge Representation System Architecture. *The European Information Society*, pages 349–364, 2007.

[43] C. Gutierrez, C. Hurtado, and A. Vaisman. Temporal RDF. In *Second European Semantic Web Conference (ESWC 2005)*, pages 93–107, 2005.

[44] R. Guting, M. Bohlen, M. Erwig, C. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Trans. Database Syst.*, **25**:1–42, March 2000.

[45] R.H. Guting. An Introduction to Spatial Database Systems. *The VLDB Journal*, **3**[4]:357–399, 1994.

[46] G. Hatzigeorgakidis. Management of Spatio-temporal Information in Semantic Web Applications. *Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete*, 2011.

[47] K. Hawley. Temporal Parts. *The Stanford Encyclopaedia of Philosophy, Edward N. Zalta (ed.)*, 2004.

[48] J.R. Hobbs and F. Pan. Time Ontology in OWL. *W3C Working Draft, September 2006*, 2006.

[49] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible SROIQ. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, 2006.

[50] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member submission*, **21**, 2004.

[51] I. Horrocks, P.F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web*, **1**[1]:7–26, 2003.

[52] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Logic for Programming and Automated Reasoning*, pages 161–180. Springer, 1999.

[53] C. Hurtado and A. Vaisman. Reasoning with Temporal Constraints in RDF. *Principles and Practice of Semantic Web Reasoning*, **4187**:164–178, 2006.

[54] C.B. Jones, A.I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The Spirit Spatial Search Engine: Architecture Ontologies and Spatial Indexing. *Geographic Information Science*, pages 125–139, 2004.

[55] P. Jonsson and A. Krokhin. Complexity Classification in Qualitative Temporal Constraint Reasoning. *Artificial Intelligence*, **160**[1-2]:35–51, 2004.

[56] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: a Declarative Query Language for RDF. In *Proceedings of the 11th international conference on World Wide Web*, pages 592–603. ACM, 2002.

[57] Y. Katz and B.C. Grau. Representing Qualitative Spatial Information in OWL-DL. In *In Proceedings of OWL: Experiences and Directions. CEUR Workshop Proceedings vol. 188*. Citeseer, 2005.

[58] M. Klein and D. Fensel. Ontology Versioning on the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, pages 75–91. Citeseer, 2001.

[59] N. Kline and R.T. Snodgrass. Computing Temporal Aggregates. In *Data Engineering, Proceedings of the Eleventh International Conference on*, page 222. Published by the IEEE Computer Society, 1995.

[60] K. Koubarakis, K. Kyzirakos, M. Karpathiotakis, C. Nikolaou, M. Sioutis, S. Vassos, D. Michail, T. Herekakis, C. Kontoes, and I. Papoutsis. Challenges for Qualitative Spatial Reasoning in Linked Geospatial Data. *Proceedings of the IJCAI 2011 Workshop on Benchmarks and Applications of Spatial Reasoning (BASR-11)*, 2011.

[61] M Koubarakis. Database Models for Infinite and Indefinite Temporal Information. *Information Systems*, **19**[2]:141 – 173, 1994.

[62] M. Koubarakis. *Foundations of Temporal Constraint Databases*. PhD thesis, National Technical University of Athens, 1994.

[63] M. Koubarakis. Temporal CSPs. *Handbook of Constraint Programming, Chapter 19*, **2**:665 – 697, 2006.

[64] M. Koubarakis and K. Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: the Model stRDF and the Query Language stSPARQL. *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010), Part I, volume 6088 of Lecture Notes in Computer Science, Springer*, pages 425–439, 2010.

[65] H.U. Krieger. A General Methodology for Equipping Ontologies With Time. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'10), ELRA*, pages 3165–3172. Citeseer, 2010.

[66] H.U. Krieger. A Temporal Extension of the Hayes and ter Horst Entailment Rules for RDFS and OWL. In *AAAI 2011 Spring Symposium Logical Formalizations of Commonsense Reasoning*, pages 143–146, 2011.

[67] H.U. Krieger, B. Kiefer, and T. Declerck. A Framework for Temporal Representation and Reasoning in Business Intelligence Applications. In *AAAI 2008 Spring Symposium on AI Meets Business Rules and Process Management*, pages 59–70, 2008.

[68] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning About Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. *Journal of the ACM (JACM)*, **50**[5]:591–640, 2003.

[69] O. Lassila, R.R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. *W3C Recommendation, World Wide Web Consortium*, 1999.

[70] N. Lopes, A. Polleres, U. Straccia, and A. Zimmermann. AnQL: SPARQLing up Annotated RDFS. *The Semantic Web–ISWC 2010*, pages 518–533, 2010.

[71] C. Lutz. Description logics with concrete domains-a survey. *In Advances in Modal Logics, volume 4. King's College Publications*, 2003.

[72] C. Lutz, F. Wolter, and M. Zakharyashev. Temporal Description Logics: A Survey. In *15th International Symposium on Temporal Representation and Reasoning, TIME'08*, pages 3–14. IEEE, 2008.

[73] X. Makri. 4D-Fluents Plug-In: A Tool for Handling Temporal Ontologies in Protg. *Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete*, 2011.

[74] E. Mavrogiannaki. The Time Dimension in Clinical Guidelines for the Management of Abnormal Gynaecological Exam Results. *Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete*, 2011.

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

[76] B. McBride. The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. *Handbook on Ontologies*, pages 51–66, 2004.

[77] D.L. McGuinness, F. Van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, **10**:2004–03, 2004.

[78] M. Minsky. Frame-system Theory. *Thinking: Readings in Cognitive Science*, pages 355–376, 1977.

[79] D.R. Montello and A.U. Frank. Modeling Directional Knowledge and Reasoning in Environmental Space: Testing Qualitative Metrics. *The Construction of Cognitive Maps GeoJournal Library*, **32**[3]:321–344, 1996.

[80] B. Motik. Representing and Querying Validity Time in RDF and OWL: a Logic-Based Approach. *The Semantic Web–ISWC 2010*, pages 550–565, 2010.

[81] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? *5th International Semantic Web Conference, ISWC 2006*, pages 501–514, 2006.

[82] B. Motik, P.F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, et al. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. *W3C Recommendation*, **27**, 2009.

[83] B. Nebel and H.J. Burckert. Reasoning About Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. *Journal of the ACM (JACM)*, **42**[1]:43–66, 1995.

[84] N. Noy, A. Rector, P. Hayes, and C. Welty. Defining N-ary Relations on the Semantic Web. *W3C Working Group Note*, **12**, 2006.

[85] T. Patkos, I. Chrysakis, A. Bikakis, D. Plexousakis, and G. Antoniou. A Reasoning Framework for Ambient Intelligence. *Artificial Intelligence: Theories, Models and Applications*, **6040**:213–222, 2010.

[86] E.G.M. Petrakis, C. Faloutsos, and K.I. Lin. Imagemap: An Image Indexing Method Based on Spatial Similarity. *Knowledge and Data Engineering, IEEE Transactions on*, **14**[5]:979–987, 2002.

[87] A. Preventis. Chronos: A Tool for Handling Temporal Ontologies in Protege. *Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete (to appear)*, 2012.

[88] E. PrudHommeaux and A. Seaborne. SPARQL Query Language for RDF. *W3C working draft*, **4**, 2006.

[89] A.K. Pujari and A. Sattar. A New Framework for Reasoning About Points, Intervals and Durations. In *International Joint Conference On Artificial Intelligence*, **16**, pages 1259–1267. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.

[90] D.A. Randell, Z. Cui, and A.G. Cohn. A Spatial Logic Based on Regions and Connection. *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, (KR 92)*, **92**:165–176, 1992.

[91] J. Renz. Maximal Tractable Fragments of the Region Connection Calculus: A Complete Analysis. In *International Joint Conference On Artificial Intelligence*, **16**, pages 448–455. Citeseer, 1999.

[92] J. Renz and D. Mitra. Qualitative Direction Calculi with Arbitrary Granularity. In *PRICAI 2004: Trends in Artificial Intelligence: 8th Pacific Rim International Conference on Artificial Intelligence, Proceedings*, pages 65–74. Springer-Verlag New York Inc, 2004.

[93] J. Renz and B. Nebel. Qualitative Spatial Reasoning using Constraint Calculi. *Handbook of Spatial Logics*, pages 161–215, 2007.

[94] J. Santos, L. Braga, and A.G. Cohn. FONTE: A Protégé Plug-in for Engineering Complex Ontologies. *Enterprise Information Systems*, pages 222–236, 2011.

[95] K. Schild. *Towards a Theory of Frames and Rules*. PhD thesis, Technische Universitaet Berlin, Berlin, Germany, 1989.

[96] A. Seaborne. RDQL-a Query Language for RDF. *W3C Member Submission*, **9**:29–21, 2004.

[97] T. Sellis. Research Issues in Spatio-temporal Database Systems. *Advances in Spatial Databases (Book Chapter)*, **1651**:5–11, 1999.

[98] N. Sepetas. Restriction Checking on OWL:2 Temporal Ontologies. *Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete*, 2011.

[99] R. Shaw, R. Troncy, and L. Hardman. Lode: Linking Open Descriptions of Events. *The Semantic Web*, pages 153–167, 2009.

[100] A. Sheth and M. Perry. Traveling the Semantic Web Through Space, Time, and Theme. *Internet Computing, IEEE*, **12**[2]:81–86, 2008.

[101] T. Sider. *Four-Dimensionalism: An Ontology of Persistence and Time (Mind Association Occasional Series)*. Oxford University Press, 2003.

[102] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, **5**[2]:51–53, 2007.

[103] S. Skiadopoulos and M. Koubarakis. On the Consistency of Cardinal Direction Constraints. *Artificial Intelligence*, **163**[1]:91–135, 2005.

[104] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems (TODS)*, **12**[2]:247–298, 1987.

[105] A. Sotnykova, C. Vangenot, N. Cullot, N. Bennacer, and M.A. Aufaure. Semantic Mappings in Description Logics for Spatio-Temporal Database Schema Integration. *Journal on Data Semantics III*, pages 143–167, 2005.

[106] M. Stocker and E. Sirin. Pelletspatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In *CEUR Workshop Proceedings*, **529**. Citeseer, 2009.

[107] U. Straccia, N. Lopes, G. Lukácsy, and A. Polleres. A General Framework for Representing and Reasoning with Annotated Semantic Web Data. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), AAAI Press*, 2010.

[108] C. Tao, W.Q. Wei, H.R. Solbrig, G. Savova, and C.G. Chute. CNTRO: A Semantic Web Ontology for Temporal Relation Inferencing in Clinical Narratives. In *AMIA Annual Symposium Proceedings*, **2010**, pages 787–91. American Medical Informatics Association, 2010.

[109] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, pages 308–322. Springer-Verlag, 2009.

[110] P. Van Beek. Approximation Algorithms for Temporal Reasoning. *Proceedings of the 11th International Joint Conference on Artificial Intelligence-Volume 2*, pages 1291–1296, 1989.

[111] P. Van Beek. *Exact and Approximate Reasoning about Qualitative Temporal Relations*. PhD thesis, University of Waterloo, 1990.

[112] P. Van Beek and R. Cohen. Exact and Approximate Reasoning about Temporal Relations. *Computational Intelligence*, **6**[3]:132–144, 1990.

[113] J. van Benthem. *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*. Springer Publishing Company, Incorporated, 2nd edition, 1991.

[114] M. Vilain and H. Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.

[115] C. Welty and R. Fikes. A Reusable Ontology for Fluents in OWL. In *Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006)*, pages 226–336, 2006.

[116] V. Zamborlini and G. Guizzardi. On the Representation of Temporally Changing Information in OWL. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International*, pages 283–292. IEEE, 2010.