Luís Miguel Barbosa Braga

Engineering Temporal and Spatial Aspects in OWL using Patterns

Luís Miguel Barbosa Braga

Engineering Temporal and Spatial Aspects in OWL using Patterns

Dissertation to obtain the degree of Master of Computer Engineering,
Specialization in Knowledge and Decision Technologies

Supervisor

António Jorge Santos Pereira

To my grandparents.

Resumo alargado

A World Wide Web (WWW) é uma rede de dados enorme, aberta, muito rica, heterogénea e não controlada. Contudo, os dados existentes na rede são principalmente orientados ao consumo humano. A Semantic Web, de acordo com a perspectiva de Berners-Lee, deve fornecer condições para que a informação publicada seja lida e interpretada/compreendida por máquinas (agentes), através do enriquecimento semântico formal e explícito. As ontologias são a especificação formal de uma conceptualização partilhada e como tal constituem um artefacto privilegiado para capturar a semântica de um modelo. O formato standard proposto pela W3C (World Wide Web Consortium) para a representação de ontologias no contexto da WWW é o OWL (Web Ontology Language).

As dimensões temporal e espacial são transversais à generalidade dos domínios. No processo de entendimento e raciocínio por agentes é crucial a consideração das dimensões temporal e espacial, em particular em tarefas como a análise de narrativas, contextualização, processamento de língua natural ou planeamento. Por exemplo, uma pessoa pode desempenhar vários papéis numa organização no decorrer do tempo; um objecto passa por diversas fases durante o processo de fabrico; ou o planeamento de uma viagem à Europa deve obedecer a diversas restrições temporais e espaciais. Apesar de os humanos demonstrarem uma capacidade inata para lidar com o tempo e o espaço, os agentes inteligentes de software precisam de especificações formais. Contudo, apesar da vasta investigação que tem sido levada a cabo no domínio da engenharia temporal/espacial esta é ainda uma tarefa complexa, trabalhosa e sujeita a erros, visto que é necessário ter conhecimento específico sobre o domínio a modelar e também sobre as teorias que modelam/capturam o tempo e o espaço.

Integrar as dimensões temporal e espacial em sistemas inteligentes é uma tarefa complexa e propensa a erros, principalmente porque:

1. muitas vezes o Engenheiro de Conhecimento tem uma percepção intuitiva e informal do tempo e do espaço, enquanto os modelos existentes são formais e complexos, resultando em sistemas nos quais não é possível explorar adequadamente estas dimensões;
2. as dimensões extra, resultantes das componentes temporal e espacial, tornam a ontologia mais complexa, aumentando a dificuldade do processo de verificação e a garantia da completude e consistência do sistema;
3. diferentes intervenientes têm diferentes percepções do tempo e do espaço.

Em particular, representar e raciocinar sobre a evolução temporal de conceitos e suas relações considerando ontologias em OWL enfrenta problemas adicionais. A linguagem OWL baseia-se na utilização de relações binárias, o que lhe confere enormes vantagens no

processamento automático mas que impõe limitações ao nível da expressividade, tornando complexo representar relações que envolvam mais do que dois argumentos (como por exemplo a caracterização temporal ou espacial de relações). A comunidade científica tem estudado várias formas para fazer face a este problema, nomeadamente:

1. extensões da Lógica Descritiva (DL) com operadores temporais e espaciais;
2. extensões do esquema formal do OWL;
3. aplicação de técnicas de gestão de versões permitindo registar o histórico da evolução da ontologia;
4. ou ainda a criação de esquemas mais complexos para a representação da informação como a criação de conceitos auxiliares para simular a existência de relações n-árias.

O principal objectivo deste trabalho consistiu no desenvolvimento de métodos e ferramentas capazes de suportar a engenharia de aspectos temporais e espaciais em sistemas inteligentes através da utilização de ontologias codificadas na linguagem OWL. Uma metodologia de engenharia de ontologias existente chamada FONTE foi utilizada como framework no desenvolvimento deste trabalho. Este método foi aplicado com sucesso na engenharia de aspectos temporais em sistemas inteligentes utilizando ontologias no formato F-Logic. O Fonte utiliza uma abordagem de dividir-para-conquistar de forma que a modelação de domínios complexos pode ser realizada através da composição de diferentes ontologias que definem as diferentes categorias de conhecimento envolvidas no domínio. O Fonte foi utilizado na engenharia dos aspectos temporais em ontologias.

Como resultado deste trabalho foi realizada a reengenharia do método FONTE de forma a suportar também a dimensão espacial e a aplicação semiautomática de padrões de desenvolvimento de ontologias (PDO). Em particular este trabalho consistiu no desenvolvimento de:

1. uma linguagem de regras que permite a implementação de PDO e a sua aplicação através da metodologia FONTE
2. mecanismos de verificação que garantem a consistência da ontologia de domínio durante o processo de engenharia;
3. mecanismos de criação automática de propostas baseados em algoritmos de pesquisa semântica e estrutural;
4. ferramenta gráfica de suporte ao método FONTE.

As capacidades da metodologia e ferramentas propostas e desenvolvidas foram demonstradas através da engenharia temporal e espacial de uma ontologia do domínio do futebol.

Abstract

WWW is a huge, open, heterogeneous system, however its contents data is mainly human oriented. The Semantic Web needs to assure that data is readable and "understandable" to intelligent software agents, though the use of explicit and formal semantics. Ontologies constitute a privileged artifact for capturing the semantic of the WWW data.

Temporal and spatial dimensions are transversal to the generality of knowledge domains and therefore are fundamental for the reasoning process of software agents. Representing temporal/spatial evolution of concepts and their relations in OWL (W3C standard for ontologies) it is not straightforward. Although proposed several strategies to tackle this problem but there is still no formal and standard approach.

This work main goal consists of development of methods/tools to support the engineering of temporal and spatial aspects in intelligent systems through the use of OWL ontologies. An existing method for ontology engineering, FONTE was used as framework for the development of this work.

As main contributions of this work FONTE was re-engineered in order to: *i)* support the spatial dimension; *ii)* work with OWL Ontologies; *iii)* and support the application of Ontology Design Patterns. Finally, the capabilities of the proposed approach were demonstrated by engineering time and space in a demo ontology about football.

Acknowledgements

There are many people that contributed to the development of this work. First I would like to thank my supervisor, Professor Jorge Santos, the guidance, support and the careful review of this document and to Professors Nuno Silva and Paulo Maio for the kindness of spending their time reviewing this document, from which resulted precious observations. Nevertheless, I will be solely responsible for any error or typo that may have subsisted.

I also would like to thank to QREN, who funded the WorldSearch Project which made possible to pursue the development of this work.

I would like to thank all my friends for the support they have shown. In particular I thank to Eduarda the support, understanding and motivation have she always demonstrated; to Nuno Fulgêncio for accompanying me during part of the development of this document; and to Nina for being able to say the weirdest words (in a good way) at the most opportune moments.

Finally, I would also like to thank my family for their understanding, affection and attention that they devote myself, specially my parents for providing me the opportunity of getting this degree.

Contents

List of Tables

List of Figures

List of Algorithms

List of SWRL Rules

Acronyms

ABox – Assertion component

DL – Description Logics

Fonte – Factorize ONTology Engineering complexity

IA – Interval Algebra

IRI – Internationalized Resource Identifier

ODP – Ontology Design Pattern

OWA – Open World Assumption

OWL – Web Ontology Language

PA – Point Algebra

RCC – Region Connection Calculus

RDF – Resource Description Framework

RDFS – RDF Schema

RIF – Rule Interchange Format

SPARQL – Simple Protocol And RDF Query Language

SWRL – Semantic Web Rule Language

TBox – Terminological component

URL – Uniform Resource Locator

URI – Uniform Resource Identifier

WWW – World Wide Web

W3C – World Wide Web Consortium

XML – eXtensible Markup Language

Chapter 1

Introduction

Suppose that someone wants to search on the Web about the football team in which Cristiano Ronaldo was playing when the UEFA European Championship took place in Portugal. He needs to perform several searches and then combine their results. First he needs to know who is Cristiano Ronaldo, in which football teams he played and when. Then he needs to know when the UEFA European Championship took place in Portugal. Finally, he needs to find some intersection between the dates of the UEFA European Championship that took place in Portugal and the several football teams where Cristiano Ronaldo played. For humans this research may not seem very complex, though laborious and time consuming. However, to intelligent software agents it is a tremendous task. The World Wide Web is a huge, very rich, open, uncontrolled, heterogeneous and human-oriented system. In order o answer complex queries, software agents must be sufficiently intelligent to find the required pieces of information and then reason over them. In particular, the ability to reason about temporal and spatial aspects is a fundamental characteristic of intelligent software agents, since these dimensions are transversal to the generality of knowledge categories.

1.1 Context and Motivation

World Wide Web (WWW) has become the most important communication platform, since the number of users and available amount of data is continuously growing. According to Internet World Stats, the number of users has grown 528.1% in the last ten years, while the traffic data increased 30638% and the CISCO predictions for 2016 suggest that the global IP traffic will increase threefold over the next 5 years which they call The Zettabyte Era. This consists in a huge opportunity for automated processing of web data and the use of intelligent software agents. However, in order to make possible the automated processing

of such heterogeneous data as on the internet it is necessary do define/capture its meaning (semantic).

The Semantic Web, according to Tim Berners-Lee [Berners-Lee et al., 2001], must provide conditions to make that data readable and "understandable" by machines, enhancing it with explicit and formal semantics. In the context of computer science an ontology is a formal specification of a shared conceptualization and as such constitute an privileged artifact for capturing the semantics of a model [Gruber, 1993]. The actual standard format recommended by the World Wide Web Consortium[1] (W3C) to represent ontologies in the web context is OWL2[2] (Web Ontology Language). The use of ontologies has grown in the last years. Projects like DBPedia[3] or YAGO[4] aim to publish the existing knowledge in Wikipedia in a structured and semantically explicit way, by using ontologies.

Temporal and spatial dimensions are transversal to the generality of knowledge domains and therefore they are fundamental for the reasoning process, as narrative analysis, contextualization, natural language processing or planning/scheduling [van Harmelen et al., 2007, Stock, 1997, Fisher et al., 2005]. For instance, a person could play several roles in an organization during time; an object passes through several phases during its manufacturing process; or planning a trip to Europe must comply with several temporal and spatial constraints. It should be noted that despite humans have developed the ability to easily deal with those kinds of scenarios, machines (*e.g.*, intelligent software agents) require formal specifications.

The study of temporal and spatial theories, and how to represent and explore them in information systems, has been receiving continuous attention by the scientific community. Despite the extensive research, the temporal/spatial engineering is still a complex, error prone process and a one-off, labour intensive experience, since it requires knowledge of the specific domain area and also of the theories that capture/model the time and space dimensions. Consequently, undesired situations may occur, such as:

- the knowledge engineer does not know which concepts may be temporal/spatial and consequently those not explicitly stated behave as atemporal/aspatial;

[1] http://www.w3.org/
[2] `http://www.w3.org/TR/owl2-overview/`
[3] `http://dbpedia.org/About`
[4] `http://www.mpi-inf.mpg.de/yago-naga/yago/`

- the granularity of temporal (*e.g.*, semester, week, day, hour) and spatial (*e.g.*, area, room, building) concepts is not properly defined resulting in overly complex (or simplistic) systems;
- difficulty of ensuring consistency knowledge by detecting temporal / spatial inconsistencies according to particular domain rules (*e.g.*, allocation multiple spaces; share the same actor in simultaneous events) since there is no distinction between domain and temporal/spatial knowledge;
- coexistence of multiple unrelated concepts that aim to capture the temporal nature of certain concepts such as a person that can play multiple roles in an organization over time (*e.g.*, teacher, president, coach, musician), and should be in fact related;
- the meaning of the temporal/spatial characterization, and how it is done, is unclear and it is encapsulated in the system logic.

These conditions contribute to the development of monoliths of information, compromising the system exploration, managing, expansion and interconnection. In addition, it will be very difficult to reuse or share the codified knowledge and so all the time and work must be re-spent in order to develop a new system. As already mentioned ontologies have the ability to express semantics and are easily shareable, so they may play a key role in representing temporal and spatial knowledge, fundamental to the development of intelligent applications.

OWL ontologies tend to represent a snapshot view of the world, the state of the domain at a given instant. Consequently it is assumed that all the axioms are simultaneously true, and that is how the inference engine interprets them. If it is considered that the ontology is able to describe the evolution of concepts and their relations it has to be assumed that the ontology will contain information that should not always be interpreted as true, but only in the specified time periods. It should be noticed that this work is not about the ontology schema evolution (*i.e.*, the changes performed over the ontology due to changes in the understanding of the domain); instead it is about how to represent and reason about information that evolve through time and space. Due to OWL characteristics, representing the temporal evolution of concepts and their relations it is not straightforward since there is no generally accepted mechanism for doing that. OWL is based on binary relations, which gives it enormous advantages in the automatic processing, in particular in the

serialization process fundamental in heterogeneous environments such as Web. However, when is necessary to add information to a relation (*e.g.*, specify that a relation holds during some period of time or that is valid only in some specific area) it must becomes ternary or N-ary, which is not supported in OWL.

The use of Design Patterns (proven modeling solutions for recurring design problems) has been crucial for the evolution of Engineering, which includes the Software Engineering and, of course, Ontology Engineering. Several Ontology Design Patterns (ODP) have been proposed to approach ontology modeling issues, namely the problem of representing temporal relations. The most common solutions to overcome this problem (*e.g.*, the N-ary relations or the 4D-fluents approach) imply the creation of new concepts to represent the temporal manifestation of the relation, unfolding a N-ary relation into a set of binary ones. The adoption of these strategies result in a more complex ontology (and possible harder to understand by humans) and it could be difficult to ensure the semantic restrictions of the original domain ontology. The main challenge in applying ODPs over domain ontologies in order to model the temporal and spatial dimensions is to ensure a balance between the ontology expressiveness and decidability (ability to be used automatic reasoning mechanisms with guarantees of success). Despite the efforts, this remains a difficult task mainly because:

- the engineering process requires the consideration of several ontological issues (*e.g.*, primitives, density, granularity, direction) often implying a complex tradeoff between expressiveness and decidability;
- the domain experts often have an intuitive and informal perception of time and space, whereas the existing models of time and space are complex and formal;
- the temporal and spatial components introduces an extra dimension of complexity in the reasoning process, making it difficult to ensure system completeness and consistency and to infer new knowledge;
- the changes that need to be performed over the domain ontology can lead it to an inconsistent state, requiring additional modifications.

Besides the previously described techniques, there are other strategies focused in the temporal and spatial ontology representation problem, such as extensions to Description

Logics (DL) with temporal and spatial operators, extension of OWL formal schema and versioning techniques. These techniques will be presented further in this document.

The problem of engineering temporal aspects in intelligent systems was address by Jorge Santos [Santos, 2008] under his Ph.D. In his work he developed FONTE[Santos and Staab, 2003a,b], a semi-automatic ontology engineering approach that uses a divide and conquer strategy to assist the user in the engineering of temporal aspects on domain ontologies written in F-Logic, by reusing temporal knowledge also modeled in ontologies. The temporal engineer is done by linking concepts of the domain ontology with concepts of the temporal ontology which will trigger the execution of Assembly Tasks. These Assembly Tasks describe which modifications should be performed to ensure the correct temporal modeling of the domain ontology concept while ensures the ontology consistency by performing verification algorithms. The FONTE method was able to assist the ontology engineering by suggesting the realization of Assembly Tasks.

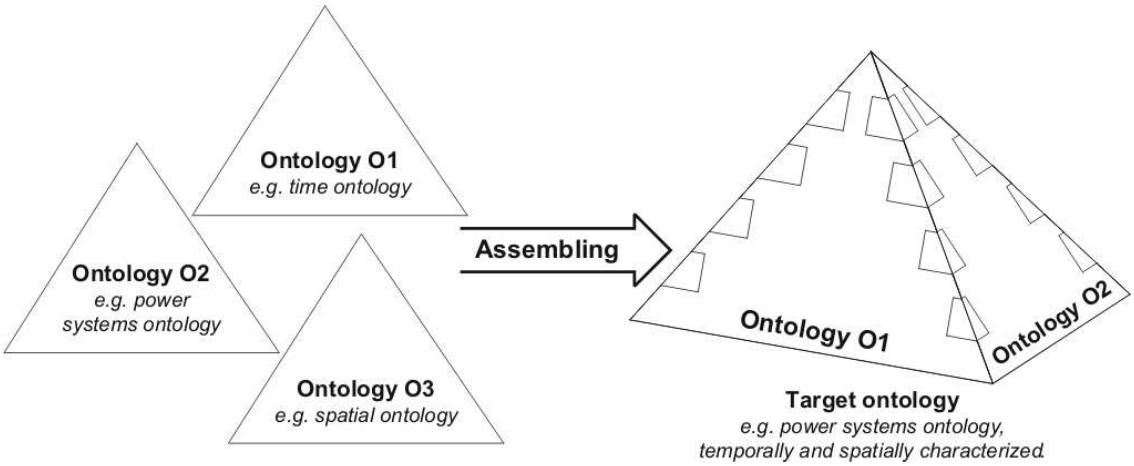


Figure 1.1: FONTE – Factorize ONTology Engineering complexity

FONTE method foresees the use of ontologies of several transversal domains, such the spatial dimension; however it was only tested with temporal ontologies. The final result is consistent multi-dimensional domain ontology, as depicted in 1.1. An text-based prototype of FONTE was developed in Prolog.

The main objective of the work described on this document consists on the study and development of methods and tools to support the engineering of temporal and spatial aspects in intelligent systems in the context of Semantic Web, by using OWL ontologies. The resulting ontologies must be capable of representing the temporal and spatial evolution

of the information (*i.e.*, the concepts and their relations) and to support the use of generic inference mechanisms for reasoning about time and space.

In particular, this work consists in two major objectives:

1. **Temporal and Spatial Representation and Reasoning in OWL**
   (a) to study and develop methods which allows the representation and reasoning about information that evolves through time and space, considering the intrinsic OWL characteristics/limitations. This implies the study of the temporal and spatial theories, namely their topology and available algebras for reasoning about temporal and spatial relations, and how to properly model them in OWL;
   (b) to study how the engineer of temporal and spatial dimensions affect the semantics of the OWL model itself, namely the class and property disjunction, and transitive, functional, inverse and symmetric properties, and how these aspects can be properly modeled.
2. **Re-engineering of FONTE method**
   (a) to re-engineer the FONTE method in order to support OWL Ontologies and also the spatial dimension;
   (b) to develop algorithms that support the specification and subsequent semi-automatic application of ODP;
   (c) to develop mechanisms that ensure the consistency of the domain ontology during the FONTE Assembly Process (*e.g.*, class or property disjointness);
   (d) to develop new methods to support the automatic generation of proposals by exploring the OWL model semantic characteristics (*e.g.*, constructs for class hierarchy or class equivalence specification);
   (e) to develop techniques to detect and filter invalid system proposals (which can arise during the assembly process) recurring to verification techniques;
   (f) to develop a graphic tool to support the FONTE method.

1.2 Contributions

Regarding the first objective of this work, a study about the temporal and spatial theories was conducted, with particular attention to their representation and reasoning using OWL

and how to consistently engineer that knowledge in domain OWL ontologies. Considering the temporal and spatial dimensions as a huge impact on how the semantic of the OWL model should be interpreted, several ODP were developed, namely how to temporally and spatially characterize classes, relations and properties according to the endurantist approach, while ensuring the ontology consistency.

In this work is also shown how some OWL notions such class and property disjunction, and transitive, functional, inverse and symmetric properties can be interpreted and modeled in order to ensure the achievement of the correct reasoning results when considering the temporal and spatial dimensions. All these modifications require several structural changes and so ensuring the ontology consistency and ensuring that the correct results are obtained through inference processes involve very considerable effort from the user.

The second objective of this work aims to tackle such difficulties, by re-engineering the FONTE method in order to make it capable of working with OWL ontologies and also consider the spatial dimension. In particular, the following changes/improvements have been implemented:

Assembly Task – the original FONTE method consisted on performing link operations between concepts of the domain ontology and concepts of the temporal ontology. In the re-engineered FONTE is possible to create and execute any kind of Assembly Task;

Assembly Rule Language – an independent rule language to specify Assembly Tasks was developed, enabling the implementation of complex ODPs. This language allows to quickly and easily developing ODPs to support different theories and ontologies of time and space. Moreover, this language supports user interaction at runtime allowing the user to guide the assembly process according to his needs;

Assembly Function – every action that should be performed during an Assembly Task execution that aims to modify the target ontology (*e.g.*, create an isA relation between classes; or remove a class restriction) is done by invoking Assembly Functions. Each Assembly Function is responsible to ensure that all pre-conditions are satisfied and that the ontology will remain consistent after its execution. The use of function frees the user from the need to know the underlying details of the ontology handling API that is being used when creating Assembly Rules by providing an abstraction

layer;

Assembly Proposal – different strategies to generate proposals of Assembly Tasks were developed. This strategies includes structural and semantic analysis of the participant ontologies and considers the OWL semantic model;

Assembly Proposal List Consistency – it was observed that the list of proposed Assembly Tasks tends to contain obsolete or incorrect proposals as the assembly process runs. Hence, it was developed a mechanism that ensures the consistency of the list of proposals by automatically remove unnecessary or incorrect proposals;

FONTE Plug-in In order to support the FONTE method a graphical tool with the same name was developed. This tool consists of a Protégé plug-in, one of the most used ontology editors. Some of the most important aspects of FONTE plug-in are:

1. the semi-automatic application of complex ODPs by executing simple Assembly Tasks;
2. the ability to manage the list of proposed Assembly Tasks by filtering the proposals according to several criteria and graphically see the relations between the proposals;
3. to undo an Assembly Task execution;
4. the existence of a process log that allows to see what changes are being applied to the target ontology.

An article about FONTE [Santos et al., 2011] was published during the execution of this work. This work was developed under the WorldSearch Project (QREN 11495 WS) funded by QREN[5].

1.3 Thesis Structure

This thesis is composed by six chapters. The main topic of this work is the engineering of temporal and spatial dimensions in OWL ontologies by using ODPs.

In the first chapter the thesis main topic is briefly characterized and contextualized. The most important aspects about the temporal and spatial engineering in OWL ontologies are introduced, as well as the motivations and the objectives of its development. Furthermore, this chapter includes the work contributions and the thesis structure.

[5]http://www.qren.pt/np4/home

The second chapter is related to the state of the art and is subdivided in three main sections. In the first section is presented an analysis of the topics of temporal and spatial representation and reasoning, in the context of artificial intelligence. The principal aspects of time and space are presented including the endurantist and perdurantist modeling approaches, the choice of the basic entity and the expressiveness impact on the developed application, and the main temporal and spatial topology aspects. In the second section the topic of Semantic Web is presented by preenting its main objective and the major technologies, with particular emphasis on OWL and Semantic Web Rules since they are explored in this work. In the third section some OWL ontologies regarding the representation of time and space domains are presented. These ontologies concern the representation of the temporal and spatial aspects presented in the first section of the second chapter. The problem of representing the temporal and spatial dimensions in OWL domain ontologies is also addressed through the presentation of several approaches documented in the literature. The notion of Ontology Design Pattern (ODP) is presented as an important tool for the engineering of temporal and spatial knowledge in OWL domain ontologies.

In the third chapter is presented an approach to consistently create OWL domain ontologies that are able to represent and reason about the temporal and spatial evolution of information, following the endurantist approach. This approach consists in reuse the knowledge modeled in ontologies of time and space and then consistently engineer it in the domain ontology by using ODPs. This chapter is divided in three sections. In the first section the used ontology about time and space is introduced by presenting some of the main classes, properties and restrictions. Then, in section two, the sample domain ontology about football is presented, which will be used to support the explanation of the integration process. The section three regards the presentation of the engineering process and addresses the temporal and spatial characterization of classes, restrictions and properties. The impact of the engineering process and the changes that are needed to perform in each case are explained in detail.

In the fourth chapter is presented the re-engineered version of FONTE an iterative and interactive method and tool for temporal and spatial engineering of domain OWL ontologies that supports the automatic application of ODPs while ensuring the domain ontology consistency and promoting the knowledge reuse. The main aspects of FONTE are presented and the iterative and interactive process is explained. Additionally is presented a

running example showing how FONTE can be used to engineering temporal and spatial dimensions in OWL ontologies using both ontologies and the ODPs presented in the third chapter.

In the fifth chapter the results of the engineering process using FONTE method are presented and discussed by providing some statistical data and discussing the FONTE utility. Additionally, is discussed how FONTE can be adapt to engineer time and space by using different OWL ontologies and even considering different modeling theories (*e.g.*, perdurantist). It is also discussed the possibility of using FONTE to engineer other transversal knowledge dimensions such user rights or to support other ontology engineering operations such ontology merging, integration or mapping.

Finally, in the sixth chapter the conclusions of the work are discussed. The limitations of the engineering approach and of the developed tool are presented. Possible directions of future research are outlined.

Chapter 2

State Of The Art

This chapter is focused on state of the art in the areas directly related to the work presented in this document. Initially the topic of temporal and spatial representation and reasoning is presented, with attention to some of the key issues and solutions within the artificial intelligence. Next is presented and described the evolution of the Semantic Web, with particular emphasis in ontologies in OWL, the main artifact for knowledge representation in Semantic Web and the most related aspect to the work described herein. Finally the difficulties in representing and reasoning about the temporal and spatial evolution of information in OWL ontologies are described. The various alternatives are presented and analyzed, by discussing their advantages and disadvantages.

2.1 Temporal And Spatial Representation

Time and space are very important aspects of commonsense knowledge [Nebel, 2007] and so representing and reasoning about these concepts is therefore fundamental in Artificial Intelligence [van Harmelen et al., 2007], particularly when approaching problems and/or applications like planning, scheduling, natural language understanding, narrative analysis, contextualization, commonsense and qualitative reasoning, and multi-agent systems [Stock, 1997, Fisher et al., 2005]. Although humans have developed the ability to deal with those kinds of scenarios, in an intuitive and informal way, machines (*e.g.*, intelligent software agents) require formal specifications that can be very complex. Temporal and spatial representations require the characterization of its basic entities and the primitive relations between them [Vila and Schwalb, 1996, Stock, 1997].

The two main approaches to model spatiotemporal knowledge are 3D (endurantism) and 4D (perdurantism). The advantages and disadvantages of each approach have been ex-

tensively discussed by the scientific community [Hayes et al., 2002]; both are successfully used in the construction of ontologies of time and space [Bittner et al., 2004, C. Welty and Makarios, 2006, Milea et al., 2011, O'Connor and Das, 2011, Batsakis and Petrakis, 2011].

In the next sections the major aspects of temporal and spatial representation and reasoning will be presented. With respect to time, the choice between instants and intervals will be discussed, as well as temporal order, density and granularity. The two main frameworks for qualitative reasoning with instants and intervals will be also presented. Regarding spatial domain, the representation using points and spatial regions (similarly to time) is addressed, followed by the presentation of the main mereotopological aspects, namely spatial orientation, distance, size and shape. Finally, two frameworks for reasoning about qualitative spatial relations about region connection and orientation are presented.

2.1.1 Endurantism

According to the 3D approach entities that are situated in space and time must persist over time without losing their identity while their relations may change. For instance, consider the entity CristianoRonaldo and the property team, which is used to relate Cristiano Ronaldo to each of the teams that he represented (or actually represents). According to the endurantist approach, the entity CristianoRonaldo persists over time and has multiple temporally located team relations, namely with SportingCP between 2001 and 2003; with ManchesterUtd between 2003 and 2009; and with RealMadridCF between 2009 and 2013. Figure 2.1 depicts the result of the temporal modeling of team relation according to endurantist approach.

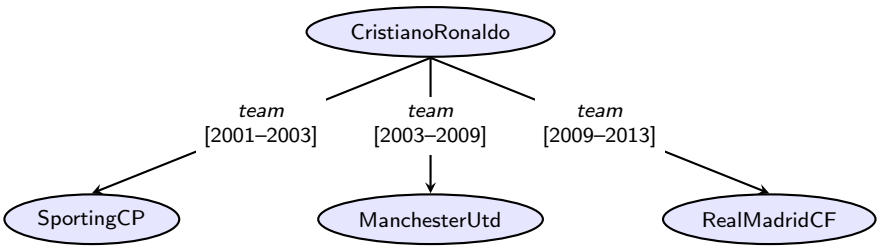


Figure 2.1: Representation of temporal evolution of Cristiano Ronaldo teams following enduratist approach

2.1.2 Perdurantism

In the perdurantist approach the entities are assumed to have distinct temporal manifestations (usually called time slices). These manifestations are located in time and space

and, for each one all relations must hold. The history of the entity existence is represented by the union of all its different time slices, as a worm is constituted by the set of its rings. Proceeding similarly with the previous example, from perdurantist point of view, the entity CristianoRonaldo would have three time slices (which are represented by CristianoRonaldo_1, CristianoRonaldo_2 and CristianoRonaldo_3, for simplicity), each one with its team relation. Each time slice has its temporal validity and its relations must hold during that period of time. In Figure 2.2 is graphically represented the result of the temporal modeling of team relation following the perdurantist approach.

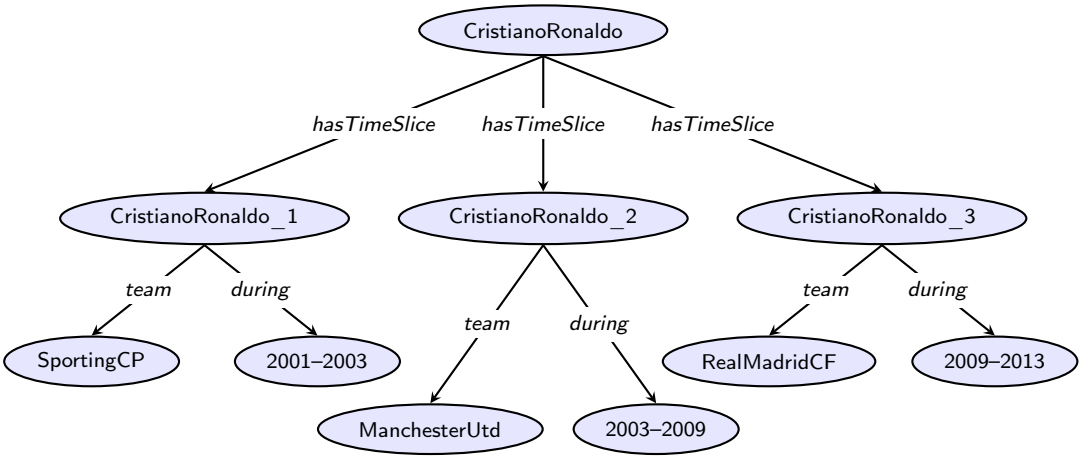


Figure 2.2: Representation of temporal evolution of Cristiano Ronaldo teams following perdurantist approach

2.1.3 Temporal Representation

The basic temporal entities are instants and intervals (also referred in literature as time points and time periods). Instants are considered infinitesimal time units of zero duration. Intervals are extended time units with start and ending time points. The advantages and disadvantages of using instants or intervals were widely discussed mainly in the 80s. Theoretically, the choice between instant-based or interval-based approaches is not crucial, because it is possible to translate one into another [Gerevini, 1997]. Although, in practice that could be an important modeling decision due to expressivity and reasoning complexity issues.

The choice of an instant-based model appears to have advantages in terms of efficiency because of its simplicity. The number of fundamental relationships between any two instants are only three (<,>,=), whereas there are thirteen in the case of interval-based models (before/after, meets/met-by, overlaps/overlapped-by, during/contains, starts/ started-by,

finishes/finished-by and equals) [Allen, 1983]. This topic is going to be explored further in this section. However, because of that an approach based on intervals provides greater expressiveness. The use of both instants and intervals in the same model seems to present some advantages and to be closer to temporal human notions. In fact, a phenomenon may be considered as an instant or as an interval depending on the perception of the modeler and the considered granularity. For instance, the 2004 UEFA Euro Championship may be modeled as an instant or, if some need to perform more complex temporal reasoning over that event, it may be modeled as an interval that started with an opening ceremony, had several games and ended with the final match and ending ceremony. In turn, these phenomena may also be considered instants or intervals. A common way of dealing with instants in interval-based approaches is to consider an instant as an interval with zero length, though this option may seem unnatural and counter intuitive.

Moreover, a temporal representation has to address other topological issues like the direction of time, density or granularity. In the linear time model there are only one time line, and time advances from past to future in a totally ordered way. It means that for each pair of different time units one precedes the other. This is the most used model in sciences like physics. In branching time model multiple (past or future) time lines are considered. It means that potentially concurrent past or future time lines are taken into account. These kinds of models are particularly useful in computer science domains as scheduling, as it allows analyzing multiple possible futures.

Relatively to density, a temporal representation is said to be discrete if assumes that between two different time units exists a finite number of other time units (or none), as happens in natural numbers. This is formally described by Axiom 2.1.1.

$$\begin{aligned} &\forall i, j, k : (i < j) \rightarrow \\ &\qquad [\exists i', j' : (i < i') \wedge (j' < j) \wedge \\ &\qquad \neg((i < k < i') \vee (j' < k < j))] \end{aligned} \tag{2.1.1}$$

On the other hand a model is considered dense if between any different time units another one exists, like real numbers. Density is formally described by Axiom 2.1.2.

$$\forall i, j \, \exists k : (i < j) \rightarrow (i < k) \wedge (k < j) \tag{2.1.2}$$

The granularity represents the unit of time representation (*e.g.*, seconds, minutes, hours, days, months). These time units can be grouped into clusters of same size and then hierarchically organized. For instance, an hour contains sixty minutes and a day contains twenty four hours. This type of representation arose from the need to organize the occurrence of recurring events in order to better understand them and to predict its occurrence and also to establish relations between different events. A calendar is an attempt to find simple mathematic relations between different granularities. The consideration of different granularities is important in the reasoning process since it allows considering the right level of detail, which have great impact both in expressiveness and performance. Usually, the smallest and nondecomposable unit of time representable in a model is termed chronon.

The temporal relations could be quantitative (if they are described using known absolute numerical values, like specific dates or measures) or qualitative (if they use lexical terms to describe the relation). Although computers deal better with quantitative relations, humans are used to describe and reason about temporal and spatial knowledge using qualitative relations. In addition, qualitative relations are a good way to deal with incomplete knowledge.

Given a set of qualitative relations, the main objectives of the process of reasoning consist of determining the consistency of the model and deduce new relationships. The most popular ways to reason about qualitative relations are by using composition tables. Initially it is necessary to define a set of basic qualitative binary relations being jointly exhaustive and pairwise disjoint. This means that between every two basic entities exactly one basic relation holds and the set of all possible relations corresponds to the set of all possible unions of the basic relations. The composition relations are the set of possible relations between two entities given its relations with a third entity. These relations are usually pre-computed and stored in a composition table, increasing the computational reasoning efficiency.

Vilain and Kautz's Point Algebra (PA) [Vilain et al., 1989] and Allen's Interval Algebra (IA) [Allen, 1983] are frameworks for reasoning about qualitative temporal relations. The PA is based on the three basic relations between instants, namely equals, before and after,

respectively denoted by the mathematic symbols $=$, $<$ and $>$. These relations are pairwise disjoint (see 2.1.3) and before and after are inverse of each other (see 2.1.4).

$$\forall i, j : (i = j \wedge i > j) \vee (i = j \wedge i < j) \vee (i > j \wedge i < j) \rightarrow \bot \tag{2.1.3}$$

$$\forall i, j : (i < j) \leftrightarrow (i > j) \tag{2.1.4}$$

The composition of any two relations of the PA is presented in the Table 2.1. The combination of before and after do not lead to a unique relation and therefore it can not be used to infer new knowledge.

Relation	$<$	$=$	$>$
$<$	$<$	$<$	$<$,=,$>$
$=$	$<$	$=$	$>$
$>$	$<$,=,$>$	$>$	$>$

Table 2.1: Composition table for point algebra temporal relations

The IA is based on the thirteen basic relations between intervals, namely before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by and equals. The symbolic and graphical representation of these relations is presented in Table 2.2, as well as the existing inverse relations. As in PA, all thirteen IA relations are pairwise disjoint.

It is possible to represent the interval relations using an instant-based approach, and vice versa. Let p_1 and p_2 be two intervals represented by their starting and ending points [sp_1,ep_1] and [sp_2,ep_2] respectively. For every interval, its starting instant always occurs before its ending instant (see Axiom 2.1.5). The thirteen Allen relations can be written as presented from Axiom 2.1.6 to Axiom 2.1.12. For simplicity, all inverse relations were omitted.

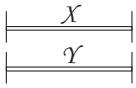
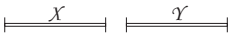
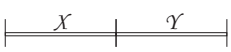
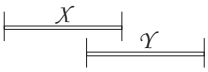
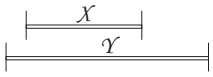
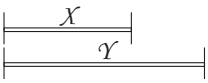
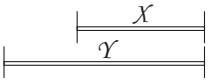
Name	Symbol	Graphical representation	Inverse symbol
equal	$=$		
before	$<$		$>$
meets	m		mi
overlaps	o		oi
during	d		di
starts	s		si
finishes	f		fi

Table 2.2: Allen's Interval Algebra relations

$$\forall p \, \exists sp, ep : (sp < ep) \tag{2.1.5}$$

$$p_1 \; equals \; p_2 \equiv (sp_1 = sp_2 \wedge ep_1 = ep_2) \tag{2.1.6}$$

$$p_1 \; before \; p_2 \equiv (ep_1 < sp_2) \tag{2.1.7}$$

$$p_1 \; overlaps \; p_2 \equiv (sp_1 < sp_2 \wedge ep_1 < ep_2 \wedge ep_1 < sp_2) \tag{2.1.8}$$

$$p_1 \; meets \; p_2 \equiv (ep_1 = sp_2) \tag{2.1.9}$$

$$p_1 \; during \; p_2 \equiv (sp_2 < sp_1 \wedge ep_1 < ep_2) \tag{2.1.10}$$

$$p_1 \; starts \; p_2 \equiv (sp_1 = sp_2 \wedge ep_1 < ep_2) \tag{2.1.11}$$

$$p_1 \; finishes \; p_2 \equiv (sp_2 < sp_1 \wedge ep_1 = ep_2) \tag{2.1.12}$$

The composition table of the basic Allen relations is presented in Table 2.3. For simplicity, the relations with equals are omitted since these compositions remains the initial relation unchanged. Only 97 out of 196 compositions leads to unique relations and therefore can be used to infer new knowledge.

	$<$	$>$	$\boldsymbol{d}$	$\boldsymbol{di}$	$\boldsymbol{o}$	$\boldsymbol{oi}$	$\boldsymbol{m}$	$\boldsymbol{mi}$	$\boldsymbol{s}$	$\boldsymbol{si}$	$\boldsymbol{f}$	$\boldsymbol{fi}$
$<$	$<$	*all*	$<$, *d*, *o*, *m*, *s*	$<$	$<$	$<$, *d*, *o*, *m*, *s*	$<$	$<$, *d*, *o*, *m*, *s*	$<$	$<$	$<$, *d*, *o*, *m*, *s*	$<$
$>$	*all*	$>$	$>$, *oi*, *mi*, *d*, *f*	$>$	$>$, *oi*, *mi*, *d*, *f*	$>$	$>$, *oi*, *mi*, *d*, *f*	$>$	$>$, *oi*, *mi*, *d*, *f*	$>$	$>$	$>$
$\boldsymbol{d}$	$<$	$>$	*d*	*all*	$<$, *d*, *o*, *m*, *s*	$>$, *oi*, *mi*, *d*, *f*	$<$	$>$	*d*	$>$, *oi*, *mi*, *d*, *f*	*d*	$<$, *d*, *o*, *m*, *s*
$\boldsymbol{di}$	$<$, *o*, *m*, *di*, *fi*	$>$, *oi*, *di*, *mi*, *si*	*o*, *oi*, *d*, *di*, $=$	*di*	*o*, *di*, *fi*	*o*, *di*, *si*	*o*, *di*, *fi*	*o*, *di*, *si*	*o*, *di*, *fi*	*di*	*o*, *di*, *si*	*di*
$\boldsymbol{o}$	$<$	$>$, *oi*, *di*, *mi*, *si*	*o*, *d*, *s*	$<$, *o*, *m*, *di*, *fi*	$<$, *o*, *m*	*o*, *oi*, *d*, *di*, $=$	$<$	*oi*, *di*, *si*	*o*	*oi*, *di*, *fi*	*o*, *d*, *s*	$<$, *o*, *m*
$\boldsymbol{oi}$	$<$, *o*, *m*, *di*, *fi*	$>$	*oi*, *d*, *f*	$>$, *oi*, *di*, *mi*, *si*	*o*, *oi*, *d*, *di*, $=$	$>$, *oi*, *mi*	*o*, *di*, *fi*	$>$	*oi*, *d*, *f*	*oi*, $>$, *mi*	*oi*	*oi*, *di*, *si*
$\boldsymbol{m}$	$<$	$>$, *oi*, *di*, *mi*, *si*	*o*, *d*, *s*	$<$	$<$	*o*, *d*, *s*	$<$	*f*, *fi*, $=$	*m*	*m*	*o*, *d*, *s*	$<$
$\boldsymbol{mi}$	$<$, *o*, *m*, *di*, *fi*	$>$	*oi*, *d*, *f*	$>$	*oi*, *d*, *f*	$>$	*s*, *si*, $=$	$>$	*oi*, *d*, *f*	$>$	*mi*	*mi*
$\boldsymbol{s}$	$<$	$>$	*d*	*b*, *o*, *m*, *di*, *fi*	*b*, *o*, *m*	*oi*, *d*, *f*	$<$	*mi*	*s*	*s*, *si*, $=$	*d*	*b*, *o*, *m*
$\boldsymbol{si}$	*b*, *o*, *m*, *di*, *fi*	$>$	*oi*, *d*, *f*	*di*	*o*, *di*, *fi*	*oi*	*o*, *di*, *fi*	*mi*	*s*, *si*, $=$	*si*	*oi*	*di*
$\boldsymbol{f}$	$<$	$>$	*d*	$>$, *oi*, *di*, *mi*, *si*	*o*, *d*, *s*	$>$, *oi*, *mi*	*m*	$>$	*d*	$>$, *oi*, *mi*	*f*	*f*, *fi*, $=$
$\boldsymbol{fi}$	$<$	$>$, *oi*, *di*, *mi*, *si*	*o*, *d*, *s*	*di*	*o*	*oi*, *di*, *si*	*m*	*oi*, *di*, *si*	*o*	*di*	*f*, *fi*, $=$	*fi*

Table 2.3: Composition table for interval algebra temporal relations

2.1.4 Spatial Representation

Similarly to time, there are two basic entities for spatial representation, namely points and extended entities (also referred in the literature as spatial regions). Computationally it is easier to deal with point-based theories which benefits from geometry and its long time research. Theories based in extended regions have been less developed but much work is currently being done, and it could be argued that they are closer to the mental and cognitive notion of space [Vieu, 1997].

Moreover, whereas the principal relations between temporal entities are based on ordering and duration, space is much more complex. This is mainly due to the higher dimensionality which enables many more different kinds of relations between its basic entities, and so much more aspects needs to be considered. The basic spatial notions often include mereotopology (relations among wholes, parts, parts of parts, and the boundaries between parts), orientation (position relatively to other spatial entity or reference point) and distance (*e.g.*, "far", "near", "further than"). According to psychological studies [Piaget and Inhelder, 1972] humans acquire these spatial notions in this particular order. Other aspects of space include size (*e.g.*, "large", "tiny", "larger than"), shape (*e.g.*, "oval", "square"), morphology and motion (spatial change) [Nebel, 2007].

Topology is the study of spatial boundaries, contact and continuity. Topological distinctions between spatial entities are a fundamental aspect of spatial knowledge [Nebel, 2007]. The classical topology theories are based on set theory, in which open or closed sets of points are represented. Despite this is a good approach for mathematics, it is often criticized for not being particularly useful to deal with commonsense spatial relations. Mereotopology emerged by the combination of topology and mereology (study of wholes, parts and their relations) as an alternative to classical topology that uses extended entities instead of points as primitive entities. Region Connection Calculus (RCC8) [Randell et al., 1992] is an algebra for reasoning about qualitative spatial mereotopological relations. The RCC8 is based on 8 basic relations between spatial regions, namely Disconnected (DC), Externally Connected (EC), Partially Overlapping (PO), Equal (EQ), Tangential Proper Part (TPP), Tangential Proper Part Inverse (TPPi), Non-Tangential Proper Part (NTPP) and Non-Tangential Proper Part Inverse (NTPPi). The topological interpretation of RCC8 relations is described in Table 2.4, in which all spatial regions are regular and closed; $i(x)$ represents the topological interior of the region x and $c(x)$

RCC8 Relation	**Topological Constraints**
x DC y	$c(x) \cap c(y) = 0$
x EC y	$i(x) \cap i(y) = 0 \wedge c(x) \cap c(y) \neq 0$
x PO y	$i(x) \cap i(y) \neq 0 \wedge c(x) \nsubseteq c(y) \wedge c(y) \nsubseteq c(x)$
x TPP y	$c(x) \subset c(y) \wedge c(x) \nsubseteq i(y)$
x TPPi y	$c(y) \subset c(x) \wedge c(y) \nsubseteq i(x)$
x NTPP y	$c(x) \subset i(y)$
x NTPPi y	$c(y) \subset i(x)$
x EQ y	$c(x) = c(y)$

Table 2.4: Topological interpretation of RCC8 relations

specifies its topological closure[1]. The graphical representation of the RCC8 relations is depicted in Figure 2.3.

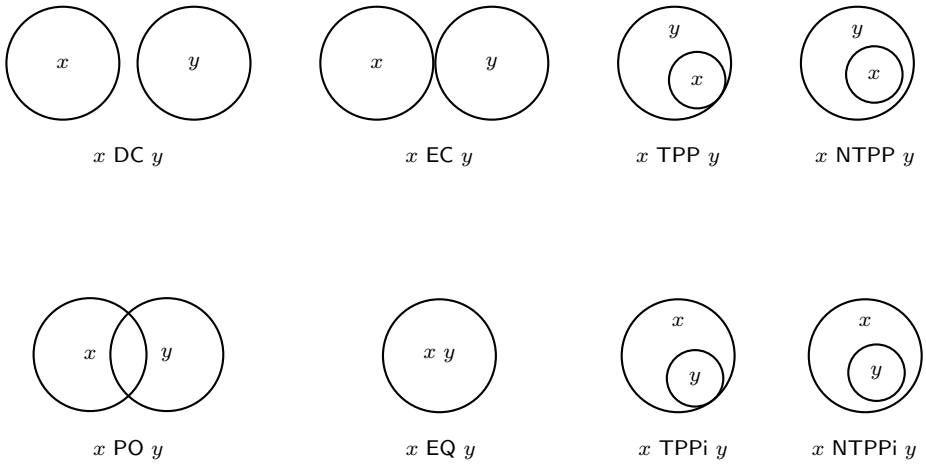


Figure 2.3: RCC8 mereotopological relations

The composition table of RCC8 relations is presented in Table 2.5. Only 27 out of 64 compositions leads to unique relations and therefore can be used to infer new knowledge.

Readers interested in further notions of mereotopology and related ontological decisions should consider [Hahmann and Grüninger, 2010].

Orientation describes in which direction a spatial entity can be found relatively to other spatial entity. The notion of orientation can be described by a ternary relation, depending on the referred object, the reference object and the frame of reference. The frame of reference could be extrinsic (if the orientation is imposed by external factors, like the cardinal direction), intrinsic (if the orientation is given by the characteristics of the reference object, like a natural front) or deictic (if the orientation is defined by the point of view

[1]The topological closure consists of all points of the region interior plus its boundary.

	DC	**EC**	**PO**	**TPP**	**NTPP**	**TPPi**	**NTPPi**	**EQ**
DC	*all*	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	*all*	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	*all*	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 2.5: Composition table for RCC8 relations

from which the reference object is seen). If the frame of reference is extrinsic, then the orientation can be described by using a binary relation, instead a ternary one.

Most of the approaches dealing with orientation assume points as primitive entities. Andrew Frank [Frank, 1991] presented different extrinsic methods to describe cardinal direction of a point relatively to some other reference point in the geographic space, namely cone-based (Figure 2.4a) and projection based (Figure 2.4b) methods. These methods allow the representation of nine different relations, namely north (N), northeast (NE), east (E), southeast (SE), south (S), southwest (SW), west (W), northwest (NW) and equal (Oc). Freksa [Freksa, 1992] developed a deictic point based approach, called double-cross calculus (see Figure 2.4c). This approach considers three axes: one must be specified according to the perspective point of the viewer and the reference point. The other two are orthogonal to the first axe and are located at the perspective point and reference point respectively. This approach results in 15 different ternary relations.

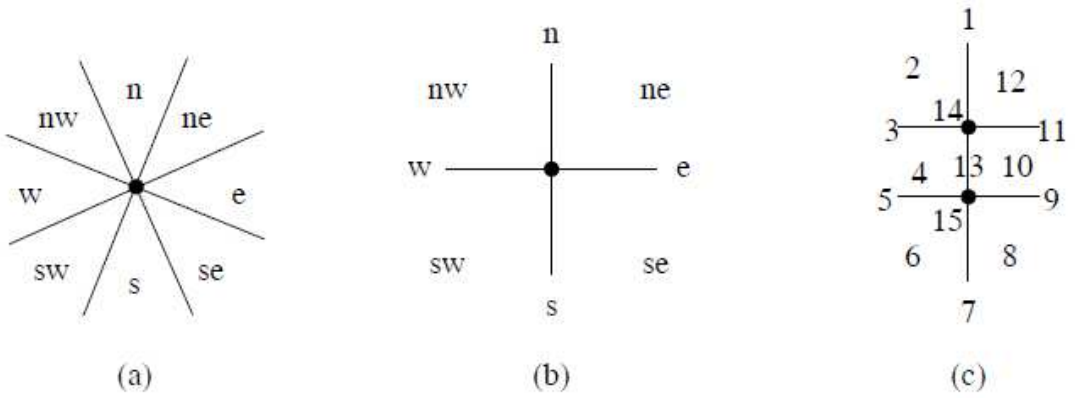


Figure 2.4: Orientation relations between points

The composition table of 2D projection-based method relations is presented in Table 2.6. Only 49 out of 81 compositions leads to unique relations and therefore can be used to infer new knowledge.

Approaches based in extended entities instead of points were also developed, despite most of them use only approximations of extended entities or consider a particular kind. This happens because is much more difficult to deal with extended entities than with points, since extended entities often have particular characteristics (intrinsic) and can represent complex shapes. It is even hard for humans to describe their relations. For instance, consider the objects A and B represented in the Figure 2.5. Even considering an extrinsic

	N	**NE**	**E**	**SE**	**S**	**SW**	**W**	**NW**	**Oc**
N	N	N	NW	E, NE, SE, Oc, N, S	Oc, N, S	W, NW, SW, N, S, Oc	NW	NW	N
NE	NE	NE	NE	E, NE, SE	E, NE, SE	*all*	N, NE, NW	N, NE, NW	NE
E	NE	NE	S	SE	SE	S, SW, SE, Oc, E, W	Oc, E, W	N, NW, NE, OC, E, W	E
SE	E, SE, NE	E, SE, NE	SE	SE	SE	S, SE, SW	S, SE, SW	*all*	SE
S	Oc, S, N	E, S, N, Oc, NE, SE	SE	SE	S	SW	SW	W, S, N, Oc, NW, SW	S
SW	W, SW, NW	*all*	S, SW, SE	S, SW, SE	SW	SW	SW	W, NW, SW	SW
W	NW	N, NW, NE, Oc, W, E	Oc, W, E	S, SE, SW, Oc, E, W	SW	SW	W	NW	W
NW	NW	N, NW, NE	N, NW, NE	*all*	W, NW, SW	W, NW, SW	NW	NW	NW
Oc	N	NE	E	SE	S	SW	W	NW	Oc

Table 2.6: Composition table of 2D projection-based method relations

referential it is not clear which orientation relation holds between the two objects.

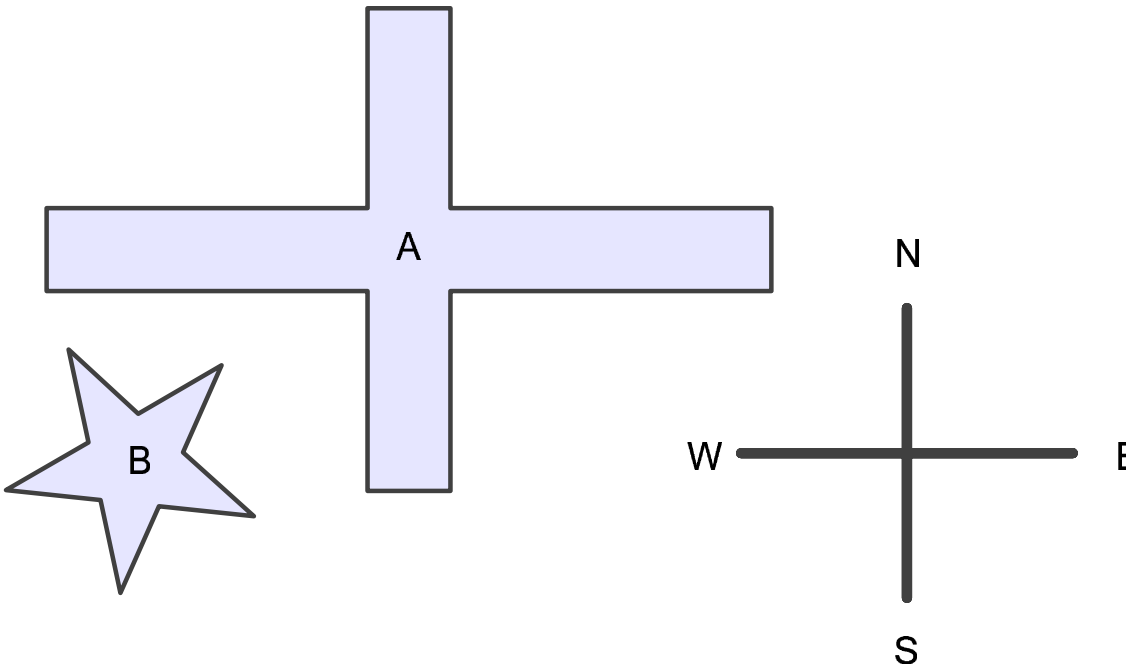


Figure 2.5: Orientation relation problem between spatial extended entities

Balbiani and colleges [Balbiani et al., 1998] developed the so called Rectangle Algebra, one approach in which extended entities are represented as rectangles parallel to the axes of an extrinsic frame of reference (see Figure 2.6a). The extended entities are represented by their projections on the frame of reference and each axe can be treated as intervals using Allen's IA. Actually the rectangle algebra can be used to combine orientation and topology relations, since it can also represent topological relations. Goyal and Egenhofer [Goyal and Egenhofer, in press] presented a calculus that consists of a 3 x 3 direction-relation matrix (see Figure 2.6b). Each sector is formed by the horizontal and vertical minimal bounding axes of the extended entity. This approach is less expressive than Rectangle Algebra since the number of possible orientation relations is smaller than in Rectangle Algebra [Nebel, 2007].

The notion of distance is present in everyday life. When the distance between two entities (extended or not) is considered then it refers to absolute distance. Absolute distance can be represented quantitatively (*e.g.*, A is twenty meters away from B) or qualitatively (*e.g.*, A is close to B), and it always depends on the used scale of space. When the distance between two entities is compared to a third, or to the distance between a third and a fourth entities, it refers to relative distance. Relative distance can only be qualitative and is represented

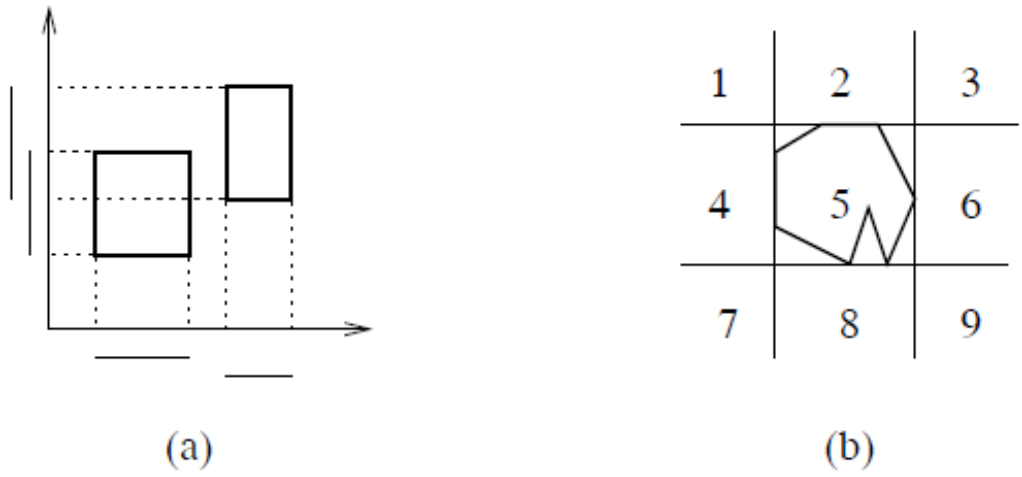


Figure 2.6: Orientation relations between extended entities

by ternary (*e.g.*, A is closer to B than to C) or quaternary relations (*e.g.*, A is closer to B than C is closer to D). To reason about qualitative distances is not straightforward. Without any further information, it is not possible to infer a distance relation between two entities by analyzing the distance relation between those entities with a third (as required to create composition tables). For instance, if someone states that A is close to B and B is away from C nothing can be concluded relatively to the proximity between A and C, since they can be close of far away from each other. Orientation can provide useful information when reasoning about distances. The combination of orientation and distance is called positional information. Clementini and colleges [Clementini et al., 1997] proposed an approach that combines cone-based orientation method and absolute distance relations. Isli and Moratz [Isli and Moratz, 1999] developed a calculi that combines relative distances with projection-based approach or double-cross calculus.

2.2 Semantic Web

In the last twenty years, the fast growth of the WWW has established a knowledge sharing infrastructure, increasing the importance of Knowledge Engineering [Studer et al., 2004]. However, most of the information available on the internet is oriented for human interaction. The knowledge is usually represented in natural language, which is suited for humans but very complex to intelligent software agents. In order to make possible for software agents to capture and properly use the knowledge available on internet efforts have been gathered in the development of Semantic Web. The goal of Semantic Web is to provide

meaning to the data. Actually, this is not a new idea or vision, Berners-Lee has presented it in 1994, at the first World Wide Web Conference, and later with an article about it in 2001, with Hendler and Lassila [Berners-Lee et al., 2001]. In that article they highlighted the potentialities of a web enriched with semantics and capable of being explored by machines that will be able to search for the right data in a particular context. This article was later revised by Hall, Shadbolt and Berners-Lee himself [Shadbolt et al., 2006]. Berners-Lee always considered that intelligent software agents would play a crucial role in the Semantic Web. In fact, he defended that the real power of Semantic Web will be realized when people create programs to analyze, collect, process and exchange the Web content automatically. The Semantic Web could be viewed as the current web increased with explicit semantics of the data and delivered in a machine readable format. It will be possible that intelligent software agents browse the internet looking for the right piece of information they need to fulfill the purpose for which they were developed. The architecture of Semantic Web is presented in the Figure 2.7, which was based on the Semantic Web Stack described by Berners-Lee [Berners-Lee, 2000].

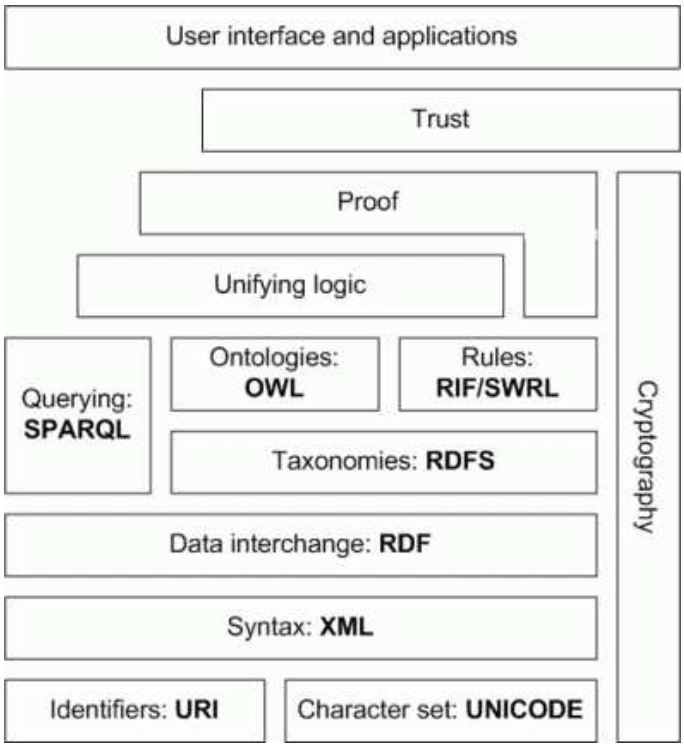


Figure 2.7: Semantic Web architecture in layers

The first layer consists of Internationalized Resource Identifiers (IRI) and Unicode character set. These are important characteristics to relate the Semantic Web with the current

human oriented WWW (usually referred as Web 2.0). Unicode is a standard of encoding international character sets and it allows that all human languages can be used (written and read) on the web using one standardized form. An IRI is a generalization of Uniform Resource Identifier (URI) and consists of a string of characters used to identify resources in the WWW. IRI is usually confused with Uniform Resource Locator (URL) because URL shares the same structure of IRI. Actually, URL is a specialization of URI (and therefore also of IRI). However, while an URL is the location of a physical resource in the WWW, the IRI does not have to point to a physical resource. Instead it can be used to describe a concept that can or can not be physically represented on the web.

The syntax layer consists of technology to encode and distribute documents over the web. The Extensible Markup Language (XML) is the standard for encoding documents and provides means for specifying and serializing structured documents which can be parsed across operating systems. Actually it is the standard syntax for knowledge encoding within the Semantic Web. Other syntax technologies such Notation 3, which is a non-XML and more user friendly notation style, has been submitted to the W3C and are waiting for approval.

The Resource Description Framework (RDF)[2] is the standard model for knowledge representation and interchange over the Semantic Web. It enables to represent knowledge in the form of graphs, by describing the binary relations between concepts in the form of Subject, Predicate, Object.

RDF Schema (RDFS)[3] provides a mechanism to describe taxonomies of classes and properties and other ontological constructs. RDFS defines formal semantics allowing the use of inference mechanisms. Both RDF and RDFS will be discussed further in this section. The Web Ontology Language (OWL) was build over the RDFS providing a more formal and restrict semantics. Actually it is the W3C recommendation for creating ontologies in the context of the Semantic Web. It derived from description logics, and offers more constructs and greater expressivity than RDFS. OWL will be presented in section 2.2.1.

Rules are used to provide a richer expressivity than the one offered by RDFS or OWL languages and so increase the reasoning capabilities. Semantic Web Rule Language (SWRL) is a rule language created as an extension to OWL. Rule Interchange Format (RIF) was

[2] `http://www.w3.org/RDF/`
[3] `http://www.w3.org/TR/rdf-schema/`

designed as an interchange format for exchanging rules between rule systems. Some misconceptions are usually taken about the differences and similarities between RIF and SWRL. Both technologies will be discussed in section 2.2.2.

Simple Protocol And RDF Query Language (SPARQL) is a SQL-Based query language to query RDF documents. Since SPARQL works at RDF triple level, it ignores the semantic of RDFS, OWL or Rules potentially encoded in RDF documents. Some SPARQL extensions such SPARQL-DL [Sirin and Parsia, 2007] has been developed to allow querying OWL ontologies while exploring their semantics.

Cryptography, Unifying Logic, Trust and Proof layers are about the need to know if the accessed information is reliable, accurate and trustworthy. The user interface and applications layer concerns the development of technology that will enable humans and intelligent software agents to explore the Semantic Web capabilities. These aspects are out of the scope of this work and so they will not be discussed in this document.

In order to make possible for intelligent software agents to process the knowledge available on the internet is necessary to provide it in a machine readable way. This does not imply that machines will be able to understand the meaning of the data they are processing, but instead they will be able to process it automatically. For instance, consider the following excerpt from the Wikipedia about the Portuguese football player Cristiano Ronaldo:

> "Cristiano Ronaldo dos Santos Aveiro, (born 5 February 1985), commonly known as Cristiano Ronaldo, is a Portuguese footballer who plays as a forward for Spanish La Liga club Real Madrid and who serves as captain of the Portuguese national team. Ronaldo was born in Santo António, a neighbourhood of Funchal, Madeira, the youngest child of Maria Dolores dos Santos Aveiro, a cook, and José Dinis Aveiro, a municipal gardener. He has one older brother, Hugo, and two older sisters, Elma and Liliana Cátia. His great-grandmother Isabel da Piedade was from Cape Verde."

This piece of text in natural language contains much knowledge about the life of Cristiano Ronaldo, including his full name, birthday, birth place, work and some family relations. However, that knowledge is not reachable by machines if it is not structured in some machine readable way.

There are many ways to organize and structure data in information systems, such as into tables (as in relational databases) or hierarchies (such in XML). However, in heterogeneous domains like WWW it is required some flexibility that allows representing any kind of information while ensuring that it could be equally interpreted by any machine. To this purpose the information contained in the excerpt is best expressed as a graph (a set of pairwise relations between nodes), as there is the idea that this kind of knowledge is often best understood as a set of concepts that are related to one another. A semantic network is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. Computer implementations of semantic networks were first developed for artificial intelligence and machine translation, but earlier versions have long been used in philosophy, psychology, and linguistics [Sowa, 2006].

In Figure 2.8 is presented a graphical representation of part of the knowledge included in the previous excerpt, encoded as a semantic network.

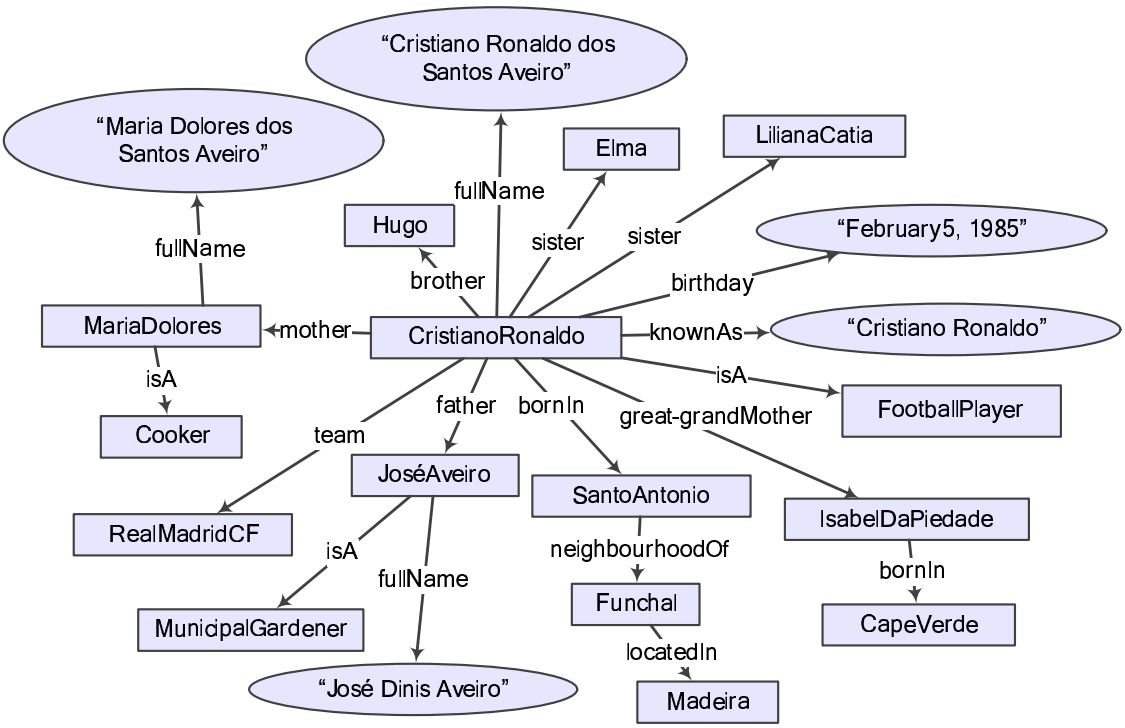


Figure 2.8: Graphical representation of knowledge about Cristiano Ronaldo encoded as a semantic network

Again, this graphical representation is good for humans, but not for machines. Instead, this same information must be represented in some machine readable way. A semantic network can be described by some textual representation. In Table 2.7 is presented a tabular representation of the semantic network from the Figure 2.8. The first and last

columns represent the start end nodes respectively, while the middle column represents the kind of connection between the two nodes.

Start Node	**Edge Label**	**End Node**
CristianoRonaldo	knownAs	"Cristiano Ronaldo"
CristianoRonaldo	fullName	"Cristiano Ronaldo dos Santos Aveiro"
CristianoRonaldo	birthday	"February 5, 1985"
CristianoRonaldo	isA	FootballPlayer
CristianoRonaldo	sister	Elma
CristianoRonaldo	sister	LilianaCatia
CristianoRonaldo	bornIn	SantoAntonio
SantoAntonio	neighbourhoodOf	Funchal
Funchal	locatedIn	Madeira
CristianoRonaldo	father	JoseAveiro
JoseAveiro	fullName	"José Dinis Aveiro"
JoseAveiro	isA	MunicipalGardener
CristianoRonaldo	mother	MariaDolores
MariaDolores	fullName	"Maria Dolores dos Santos Aveiro"
MariaDolores	isA	Cooker
CristianoRonaldo	greatGrandMother	IsabelDaPiedade
IsabelDaPiedade	bornIn	CapeVerde
CristianoRonaldo	team	RealMadridCF
CristianoRonaldo	brother	Hugo

Table 2.7: Tabular representation of the semantic network

The necessity to represent the knowledge modeled by semantic networks in a machine readable way lead to the development of RDF.

RDF is a standard and a W3C recommendation for describing knowledge in the context of Semantic Web. Created in 1999 for encoding metadata about web resources (such as the author or creation date of a web page), the RDF is now used to encode knowledge about real-world things and their relationships. The purpose of RDF is to be a flexible and simple way to express any fact and yet so structured that computers are able to use that information properly.

RDF Model specifies a formal way to decompose knowledge into small pieces. Each of these pieces is called a statement or triple in the form Subject, Predicate, Object (S,P,O). Each part of the triple has its own meaning: the Subject and the Object are names for things in real life and correspond to the nodes of a graph; the Predicate is the name of the relation that holds between the Subject and Object, and corresponds to the edge of the graph.

RDF resources must be represented using Internationalized Resource Identifiers (IRI) which are universally unique. That means that two persons can not use the same IRI to denote different concepts. IRIs are often represented using qualified names[4] (QName), which consist of declaring a small word (prefix) to describe the namespace and then use it along with the local name. For instance, the IRI `http://www.fonte-project.net/ontologies/footballExample#CristianoRonaldo` can be divided in namespace (`http://www.fonte-project.net/ontologies/footballExample#`) and local name (CristianoRonaldo). By declaring that the word "ex" corresponds to the former namespace, the IRI can be represented by the QName ex:CristianoRonaldo. The Notation 3 syntax explores the use of QNames.

RDF allows users to describe real world things and their relations in a formal way, and computers to interpret them. However, using only RDF, computers will not be able to perform inference (to derive logical conclusions by analyzing a set of evidences).

RDFS increases the RDF expressiveness power by providing formal definitions for inference. RDFS is a W3C recommendation since February 10, 2004. For more information about RDFS, the reader may consult the W3C official RDF Schema documentation[5].

The main aspects of RDFS are the notions of class, property, individual and relation. Classes describe abstract concepts and can be used to organize sets of individuals with similar characteristics. Additionally classes can be organized in hierarchies by using the property rdfs:subClassOf. If a class is defined as sub class of another, then all the individuals of the first also becomes individuals of the later. For instance, if the class ex:FootballPlayer is a sub class of ex:Person and the individual ex:CristianoRonaldo is of type ex:FootballPlayer, then ex:CristianoRonaldo is also of type ex:Person. This property is transitive (see Axiom 2.2.1), which means that if it is specified that some class C2 is sub class of some class C1 then it is assumed that every sub class of C2 is also sub class of C1.

$$\forall x, y, z : P(x, y) \wedge P(y, z) \rightarrow P(x, z) \tag{2.2.1}$$

Properties are used to establish relations between resources. RDFS model provides

[4] The formal definition of Qualified Names can be found at `http://www.w3.org/TR/xml-names11/#ns-qualnames`

[5] `http://www.w3.org/TR/rdf-schema/`

several properties with specific semantics that can be explored in the inference process. The property rdf:type is used to assign an individual to a class. For instance, if ex:CristianoRonaldo is of rdf:type of ex:FootballPlayer then the inference process will conclude that ex:CristianoRonaldo is an individual, ex:FootballPlayer is a class and that ex:CristianoRonaldo belongs to the class ex:FootballPlayer.

Similarly to classes, is also possible to describe property hierarchies by using the rdfs:subPropertyOf property. If a property is sub property of another, then all resources (classes or instances) related through the first are also related through the later. This property is also transitive (see Axiom 2.2.1).

Another two useful properties are rdfs:domain and rdfs:range, which are used to provide more information about properties. The domain and range of a property specify that the subject and object of a relation through that property must be classified according to the defined domain and range. There are two other properties that are used to provide human-readable content, namely rdfs:label (to specify the resource name) and rdfs:comment (to register a description of the resource).

RDFS defines the semantic for other classes such rdfs:Container (to describe open collections) and rdfs:Collection (to describe closed collections) and for their related properties. Since their semantics will not be explored in this work, a description about them will not be provided in this document.

There are parts of RDF and RDFS vocabularies which semantics is not formally specified, in particular concerning the use of reification and containers. The way to interpret this vocabulary is often adapted to user needs and so it can not be used consistently over the world. The need for a better defined semantics and more expressive power led to the development of OWL.

2.2.1 OWL – Web Ontology Language

Web Ontology Language (OWL) is the successor of RDFS vocabulary. It increases RDFS expressive power by defining a richer vocabulary and formal semantics.

In the context of computer science, an ontology (also referred in literature as schema or vocabulary) is a formal specification of a conceptualization about a specific portion of the world [Gruber, 1993]. Latter, Studer et al. proposed a redefinition of the concept of

ontology by including the notions of sharing and explicitation. Ontology is an explicit and formal specification of a shared conceptualization. Its main purpose is to provide formal representations of models that can be easily shareable and understandable both by humans and machines [Studer et al., 2004].

Ontologies are often divided in two components: Terminological component (TBox) and Assertion component (ABox). The TBox consists of logical definitions of concepts (classes), relations (properties) and the asserted axioms (restrictions). The ABox contains the individuals (instances of classes) and the instances of relations between them. Together they form the so called knowledge base. Making an analogy with databases, the TBox can be considered the database schema, while the ABox corresponds to the table records.

OWL has a broad and complex variety of characteristics, hence list and describe each of them would be too extensive and escapes the scope of this document. In addition to the more general concepts, the emphasis will be to the characteristics that were explored in this work. The descriptions will concern the OWL2, the current W3C recommendation for building of ontologies within the Semantic Web and direct successor of OWL. Since the similarity between OWL and OWL2 is much higher than that between RDFS and OWL, the differences between both OWL versions will not be listed here. Therefore, for now on the term OWL will be used to refer to OWL2.

Actually, one of the main differences between RDFS and OWL is the ability to represent two kinds of classes. Class descriptions allows to describe abstract concepts and can be used to organize sets of individuals with similar characteristics. OWL distinguishes between two kinds of class descriptions, namely Named Classes and Restrictions (also known as Anonymous Classes).

A Named Class is a special kind of class that is described by its name (represented by an IRI). This notion is similar to the RDFS class. The notion of Restriction will be presented later. OWL model includes special named classes with particular semantics, namely:

owl:Thing – represents the set of all things. All classes are subclasses of owl:Thing and every individuals are of type owl:Thing;

owl:Nothing – represents the empty set. No class can be subclass of and no individual can be of type owl:Nothing. This class is subclass of every class.

As in RDFS, properties are used to establish relations between resources. OWL considers three kinds of properties, namely Object Properties, Data Properties and Annotation Properties.

owl:ObjectProperty – includes the properties that can be used to establish a relation between two classes, other properties and individuals;

owl:DataProperty – includes the properties that can be used to establish a relation between an classes or individuals and a Data Range[6] (concrete domain values such integers or strings);

owl:AnnotationProperty – includes the properties that can be used to add annotations to an ontology, axiom or IRI. The rdf:label and rdf:comment properties presented earlier are classified as Annotation Properties in OWL. Since Annotation Properties are human oriented they are not explored in the inference process and so will not be further described in this document.

Like RDFS, it is also possible to define the properties domain and range. The domain and range of Object Properties and the domain of Data Properties must be a class description. The range of data property must be a Data Range.

OWL provides de ability to define the formal semantics of properties by specifying some characteristics. The available characteristics for Object Properties are:

Funtionality – it specifies that an object property is functional, which means that an individual can not be related to more than one individual through a functional property P at the same time;

Inverse Functionality – it specifies that the functionality of the property must be guaranteed in the inverse direction of the relation;

Reflexivity – it specifies that the object property is reflexive. Reflexive properties relate every individual to themselves;

Irreflexivity – it specifies that the object property is irreflexive. Irreflexive properties can not be used to relate an individual to itself;

Transitivity – it specifies that the object property is transitive. If the property P

[6]The notion of Data Range will not be discussed in this document. For further information about the use of Data Ranges in OWL please consult `http://www.w3.org/TR/owl2-syntax/#Data_Ranges`

is transitive, individual I1 is related to I2 through P and I2 is related to I3 through P then I1 is also related with I3 through P;

Symmetry – it specifies that the object property is symmetric. Symmetric properties are also applied in the inverse direction of the relation. If a property P is symmetric and an individual I1 is related to I2 through P then I2 is also related to I1 through P;

Asymmetric – it specifies that the object property is asymmetric. Asymmetric properties can not be applied in both directions of the relation. If a property P is asymmetric and I1 is related to I2 through P then I2 can not be related to I1 through P;

The functionality is also available for Data Properties and it specifies that if a data property is functional, which means that an particular individual can not be related to more than one different literal value through a functional data property P at the same time.

A Restriction or an Anonymous Class is a class that is described by the constraints that must be satisfied in order for an instance to belong to them. OWL provides several logical operators to describe membership conditions, namely: and ($\wedge$), or ($\vee$), not ($\neg$), universal quantification ($\forall$), existential quantification ($\exists$), number restrictions, enumeration of individuals and self-restriction. The operators $\wedge$, $\vee$ and $\neg$ are used as in classical logic. The existential class expression define that all individuals that are connected to an individual belonging to a specific class through a specific property are individuals of that class expression. The universal class expression defines that all individuals belonging to that class expression must be connected only to individuals of a specific class through a specific property. Since OWL is based on Open World Assumption (OWA), universal class expressions can not be used for automatic classification. The individual value restriction is used to define a class expression that contains all individuals that are connected to other specific individual through a particular property. It is the combination of both existential quantification and enumeration. The enumeration of individuals allows specifying that a class expression is the set of a particular group of individuals. The self-restriction allows specifying a class expression in which all individuals are connected to themselves by a specific property.

OWL also allows the use of number restrictions in combination with universal, existential

and individual value restrictions. With number restrictions it is possible to declare how much property relations an individual has to be in order to satisfy the restriction, by means of at least, exactly or at most a specific number of relations. Only at least number restrictions can be used for inference, because of OWA.

Excluding self-restriction that is allowed only with object properties, all operators can be used with both object and datatype properties. In the case of datatype properties, a specific datatype or value must be used as range of property relation instead of class or specific individual, respectively.

OWL model also provides several built-in properties that can be used to relate class descriptions and properties.

owl:subClassOf – an property already introduced when describing RDFS, which can be used to relate one class (named or anonymous) to other class from which the first is a sub class of. The owl:subClassOf property is transitive;

owl:equivalentClasses – it allows specifying that some class description is equivalent to another one. This means that if classes C1 and C2 are equivalent, then all individuals of C1 are individuals of C2 and vice-versa. This property is transitive and symmetric;

owl:disjointClasses – it allows specifying that two class descriptions are pairwise disjoint. This means that there could not be any individual that belongs to both classes at the same time. This property is symmetric;

owl:subObjectPropertyOf – this property semantics is similar to the rdfs:subPropertyOf, already introduced in the RDFS description. It can be used to establish the object property hierarchy. This property is transitive;

owl:subDataPropertyOf – similar to owl:subObjectPropertyOf but for data properties;

owl:equivalentObjectProperties – allows to describe an equivalence relation between object properties. If two properties P1 and P2 are equivalent and an individual I1 is connected through P1 to I2, then I1 is also connected to I2 through P2. This property is transitive and symmetric;

owl:equivalentDataProperties – similar to owl:equivalentObjectProperties but for data properties;

owl:disjointObjectProperties – similarly to owl:DisjointClasses it allows to specify that two disjoint properties can not be used to relate two same individuals. If two properties P1 and P2 are disjoint then can not be two individuals I1 and I2 simultaneously connected through P1 and P2;

owl:inverseObjectProperties – it allows specifying that two properties are inverse of each other. If two properties P1 and P2 are inverse of each other and an individual I1 is related to I2 through P1, then I2 is related to I1 through P2;

The OWL ABox includes the declaration of individuals and their relations. Individuals are considered the objects of the domain. There are two kinds of Individuals, namely Named Individuals and Anonymous Individuals.

NamedIndividual – those who are identified by an IRI and as such may be globally referenced;

AnonymousIndividual – those who are used to satisfy specific ontology local needs and so are not expected to be used outside of the ontology. They can be identified by using a local ID instead of a full IRI.

A relation is connection between two individuals through a property.

2.2.2 Rules

Despite OWL provide a very high expressive power it only addresses a part of the problem space, which concerns classification problems that can be expressed using Description Logic. Many users may need to augment OWL with rules that either can not be implemented with OWL or will be prohibitively difficult to do so.

The Semantic Web Rule Language (SWRL) [Horrocks et al., 2004] was developed to try to overcome this problem. SWRL is an OWL-Based rule language that allows enriching OWL ontologies with more powerful deductive reasoning capabilities by extending OWL axioms to include Horn-like clauses. SWRL rules are consisted of the antecedent part (also referred as the body) and the consequent part (also referred as the head) and, like OWL, is based on binary predicates. The generic formula is as presented in the SWRL Rule 2.1, where p is a predicate symbol and $var_1, \ldots var_n$ are the arguments of the expression.

SWRL does not support direct reasoning about classes or properties; it only allows mak-

$$p(?var_1, \ldots var_n) \wedge \ldots p(?var_1, \ldots var_n) \rightarrow p(?var_1, \ldots var_n) \wedge \ldots p(?var_1, \ldots var_n)$$

SWRL Rule 2.1: SWRL generic formula

ing inferences about individuals. One of the most powerful features of SWRL it is the ability to support built-ins. Built-ins are special predicates that can take one or more arguments in order to perform specific tasks (*e.g.*, value comparison, math calculus or string manipulation). A complete list of the SWRL core built-ins can be found at `http://www.daml.org/rules/proposal/builtins.html#8.1`. SWRL also provides some experimental built-ins which include a built-in (swrlx:makeOWLThing) capable of creating OWL individuals at runtime. Additionally, SWRL allows the user to create his own built-ins and use them in rules.

When considering all of the features offered by SWRL it is not possible to ensure the ontology decidability. However a subset referred as DL-Safe SWRL rules ensures decidability and is actually supported by the majority of the OWL reasoners. According to the SWRL official FAQ webpage[7]:

> "DL-Safe SWRL rules are a restricted subset of SWRL rules. These rules have the desirable property of decidability. Decidability is ensured by restricting rules to operate only on known individuals in an OWL ontology. More precisely, all variables in a DL-Safe SWRL rule bind only to known individuals in an ontology."

For that reason, built-ins such that capable of creating OWL individuals at runtime can not be used in DL-Safe SWRL rules and are not supported by the majority of the reasoners. One solution to overcome this problem is using SWRL rules combined with Jess Back-End, a powerfull rule engine for the java which supports the use of this built-in. However, Jess Back-End is a commercial application, which can be a problem for many users.

Rule Interchange Format (RIF) objectives superseeds the objective of SWRL. Its main purpose is to allow different rule languages to be interchanged across the WWW and understandable by any rule engine. For instance, someone could create a rule according to the rule language A, translate them to RIF, publish them on the web and someone

[7] `http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ#nid9VC`

else would be able to translate them to the rule language B in order to be executed in a different rule engine. Readers interested in translating rules from and into RIF may consider reading [Ma and Wang, 2012].

Different rule languages have different expressiveness capabilities and so design a single format that supports all existing languages is not easy. For that reason RIF provides multiple versions, called dialects. The standard RIF dialects are Core, BLD, and PRD. Apart from being a format for rule interchange, each RIF dialect is also a standard rule language and so it can be used to create rules from scratch.

RIF is not directly related to OWL, at least not as much as SWRL. However, taking into account the importance of the RDF, RDFS and OWL in the context of semantic web, the RIF Working Group as defined the RIF-RDF and OWL Compatibility.

The work described in this document considers SWRL rules to increase the OWL reasoning capabilities. Although SWRL is still not a W3C recommendation it already has several years of development, has a large user community, good documentation and is highly supported by several ontology editing softwares and inference engines.

2.3 Time and Space in OWL Ontologies

In recent years, different ontologies about time and space have been developed and are now available in the public domain: specific ontologies about time and/or space like OWL-Time[8], SWRL Temporal Ontology[9], SWEET-Time and SWEET-Space[10]; and upper ontologies (also called general) that include components describing time and/or space like SUMO[11], OpenCYC[12], GUM[13] and more recently COSMO[14], which is an effort to create a higher upper ontology by merging several others upper ontologies and by developing some specific concepts. More than providing methods for reasoning over time and space, these ontologies concern the representation of temporal and spatial aspects. Although representing temporal concepts in OWL may not be a difficult task, representing relations that evolve through time it is not trivial.

[8] www.w3.org/TR/owl-time
[9] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalOntology
[10] http://sweet.jpl.nasa.gov/ontology
[11] www.ontologyportal.org
[12] www.opencyc.org
[13] www.ontospace.uni-bremen.de/ontology/gum.html
[14] http://micra.com/COSMO/

For instance, consider a contract between a football player and a team, represented by the binary relation *playerOf*(FootballPlayer,Team). In real life it is reasonable to say that this relation can incur in changes through time and the player can be no longer playing for that team in the future (and even play again later). This could be done by adding a new parameter to the relation, representing time, converting it to a ternary relation. Hence, the relation could be updated to *playerOf*(FootballPlayer,Team,Time). The same could be said for spatially restricted relations. For instance, the court maximum sentence of a country could vary by geographic location. Since OWL only supports binary predicates it is not possible to simply add a new parameter to the relation. Several approaches have been proposed to address this problem. These approaches could be grouped in four categories, namely standard Description Logics extensions, schema extensions, versioning techniques and user defined models.

Description Logics extensions are extensions to standard DL with additional constructs for temporal and spatial representation [Lutz et al., 2008, Krieger, 2008, Artale and Franconi, 2001, Wessel, 2002]. This approach would provide additional expressive capabilities and straight semantics and do not suffer from data redundancy. However it requires extending OWL syntax and semantics to support additional constructs.

Formal schema extensions are modifications to the original OWL schema that provides formal methods for representing temporal and spatial aspects over classes, restrictions and properties. Temporal RDF [Gutierrez et al., 2007] allows the labeling of properties with temporal information (*e.g.*, time interval in which the property holds), which implies extending standard RDF syntax and semantics. Moreover, temporal RDF does not support expressing incomplete information through qualitative relations. Temporal OWL approaches consist in extending OWL (usually a DL subset) with temporal notions and strict semantics so the reasoners will be able to process that temporal information. One example is tOWL [Milea et al., 2011], a temporal OWL extension with concrete domains, and time representation following 4D-Fluent approach. In [Hogenboom et al., 2010] the authors conclude that OWL does not have enough expressive power to represent and reason about all RCC8 relations.

Versioning techniques suggests the creation of several snapshots, each one labeled with its valid time [Klein and Fensel, 2001]. Every time something changes in the ontology, a new version is created. Sem Version [Völkel et al., 2006] is a versioning system for RDF

models and OWL ontologies. This approach has several disadvantages:

1. data redundancy, since even for small changes a new whole version of the ontology is created;
2. query performance, because queries could require exhaustive search of all versions;
3. it is not clear how relations through classes evolve.

The use of named graphs could reduce the search space. Named Graphs consists of many RDF Graphs nameable by IRIs connected to each other [Carroll et al., 2005]. Each RDF subgraph (named graph) has an associated time interval in which all relations must hold. However, named graphs are not part of OWL Specification and are not supported by OWL reasoners [Batsakis and Petrakis, 2011].

User defined models consist in developing a temporal/spatial model on top of OWL, without the need of modifying its logical model. The two most common approaches are N-ary Relations [Noy and Rector, 2006] and 4D-Fluents [C. Welty and Makarios, 2006]. N-ary relations is an approach inspired on RDF reification[15] and it consists in creating a new class to represent the N-ary relation. The new class will specify the previous relation plus the needed restrictions to represent the temporal extent. This approach can be used to model time and space according to the endurantist approach. 4D-fluents (or 4-dimensionalist) approach reflects the perdurantist view. In this approach the dynamic (perdurants) and the static (endurants) parts of the ontology are clearly separated. Temporal concepts are represented as 4-dimensional objects (usually termed time-slices), with the 4th dimension being time. Each temporal concept is represented by all his time-slices and for each time-slice all attributes must hold. Both these approaches suffer from data redundancy and have limited OWL reasoning capabilities, since the semantic of the temporal/spatial restrictions is not part of OWL. In this work are used user defined models implementations, mainly because:

1. the lack of temporal and spatial constructs implies that logic based approaches require to extend the DL and/or OWL logical model and, despite the variety of proposed extensions, there are still no agreement on a standard approach;

[15] `http://www.w3.org/TR/rdf-primer/#reification`

2. versioning techniques suffer from several disadvantages, and therefore they were not considered a viable alternative;

3. despite the known issues of using user defined models , SWRL/RIF rules can significantly improve the temporal and spatial reasoning capabilities, and can be quickly and easily implemented.

As stated earlier, there are several ontologies that represent temporal and spatial concepts; hence they can be used as building blocks to create new complex ontologies, promoting modularity, reuse and dividing the engineering process complexity. In Software Engineering, modular programming is considered a technique for software development in which functions or features are divided into different modules (or components). Each module can be related with others but have all the necessary information to fulfill its function separately. These notions can be applied to ontologies hence there is a growing interest in that topic [C. Welty and Makarios, 2006, d'Aquin et al., 2009, Bezerra et al., 2009]. Ontology modularization has many benefits, namely in design, interpretation and validation [Stuckenschmidt, 2003], maintenance, reuse [Doran, 2006] and combinations of multiple ontologies (*e.g.*, ontology merging, mapping, alignment, refinement, unification, integration, and/or inheritance).

The reuse of temporal and spatial knowledge in a consistent way is not straightforward. It can be a one-off, labour intensive and time-consuming experience that may require domain expertise. Some of these issues could be alleviated by using design patterns. Design patterns describe common solutions for recurring design problems. There are several types of ontology design patterns (ODPs), each one with a specific propose [Gangemi and Presutti, 2009]. Logical ODPs are independent of a specific domain and deal with expressivity problems of a specific language (in the context of this work, OWL). An example of a Logical OP is the N-ary Relations pattern [Noy and Rector, 2006] to solve the problem of representing N-ary relations in OWL. Content ODPs, on the other hand, are domain dependent, because they deal with modeling problems for a specific domain. The N-ary Participation pattern [Gangemi, n.d.] is an example of a content OP to represent events with their participants and temporal and spatial locations.

The process of engineering temporal and spatial notions in domain ontologies could imply several semantic, structural and logic changes in the ontology. During this process many

classes, properties and axioms could be added, related or removed, which possibly leads the ontology to an inconsistent state. Also, when considering non static ontologies the OWL semantic itself needs to be reconsidered. It is important to ensure that after the process the ontology is still consistent and at least the reasoning results and the queries that were previously made may continue to be answered. However, even small changes could affect the overall ontology and guarantee the consistency manually is a hard and error-prone task.

Chapter 3

Engineering Time and Space in OWL Ontologies

Take into account the temporal and spatial dimensions in OWL domain ontologies has a significant impact on how the information is represented, reasoned and queried.

In this chapter is shown how to engineering temporal and spatial aspects in OWL domain ontologies by reusing existing temporal and spatial knowledge. The demonstration of the temporal and spatial engineering in OWL domain ontologies will be supported by a case study in order to provide examples for a better understanding. In section 3.1 the ontology about time and space is presented. This ontology is a version COSMO ontology enriched with SWRL rules that allows implementing PA (Point Algebra), IA (Interval Algebra) and RCC8 (Region Connection Calculus 8). In section 3.2 the domain ontology is presented; an atemporal and aspatial ontology about football. Additionally, a set of domain rules are described, which will allow a clear demonstration about the importance of considering time and space in domain ontologies. In section 3.3 the process of engineering time and space in OWL domain ontologies according to the endurantist theory is described in detail. It will be explained how it is possible to temporally and spatially engineer classes, properties and relations. The new possible interpretations of OWL, resulting from the temporal and spatial consideration, are discussed and it is explained how they can be properly modeled. In particular is explained how restriction checking over temporal properties is ensured, namely on transitive, functional, inverse functional, symmetric, asymmetric and disjoint properties.

3.1 Time and Space Ontology

The COSMO ontology will be used to show how temporal and spatial notions can be engineered in OWL. For simplicity is presented herein an excerpt containing only the fundamental temporal and spatial concepts of COSMO. A graphical representation of this excerpt is shown on Figure 3.1. For now on, the prefix "cosmo:" will be used to denote entities of the COSMO ontology.

For sake of usability a module containing only the fundamental temporal and spatial concepts of COSMO was manually extracted.

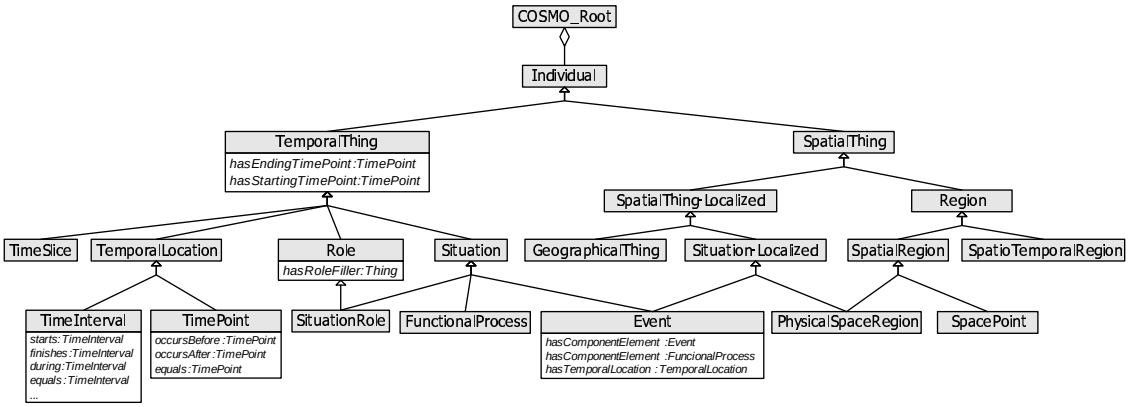


Figure 3.1: Excerpt of the COSMO ontology containing the main temporal and spatial entities

The main temporal concept is represented by the class cosmo:TemporalThing, which represents all entities that have a beginning and ending point in time. One of the most important subclasses is cosmo:TemporalLocation, which describes the temporal location for something, usually an event. This class includes the description of temporal instant and interval basic entities, through cosmo:TimePoint and cosmo:TimeInterval respectively. COSMO adopt the interpretation that a time interval of zero length duration is indistinguishable from a time point, and so cosmo:TimePoint is subclass of cosmo:TimeInterval. A cosmo:TimePoint is a closed interval of time having zero length. The start and ending time points must be identical for any given cosmo:TimePoint, and equal to the cosmo:TimePoint itself. In the absence of any explicit clock designation, every temporal location is assumed to be relative to NIST atomic clock[1]. For simplicity, the cosmo:TimePoint class was enriched with a restriction stating that every time point must have a cosmo:hasDateTime attribute of type xsd:DateTime. It will also be useful to implement the PA and IA, as it

[1]NIST stands for National Institute of Standards and Technology. More information about the NIST atomic clock can be obtained at `http://nist.time.gov/`

will be shown later in this section.

The class cosmo:TimeSlice is used to represent any temporal part of some entity on a 4D (perdurantist) treatment of objects. Every timeslice has starting and ending points and, it exists during all that period of time and all its relations hold.

Temporal Roles are represented by the class cosmo:Role, since it is a subclass of cosmo:TemporalThing. Despite the difficult to clearly define what a role is, in this work we interpret a role as a part played/performed by an entity during some period of time, which seems to be closely related to the definition provided by the COSMO authors:

> "Role is a high-level concept that aggregates several primitive notions, and is difficult to describe analytically, but has a necessary property that, as a subtype of TemporalThing, every instance has a beginning time and an ending time. (...)"

Events in COSMO, represented by the class cosmo:Event, are assumed to be things that happen during some period of time and at some particular spatial location, and so the class is subclass of both cosmo:TemporalThing and cosmo:SpatialThing. Those events can potentially affect the state of its participants and be constituted by a group of other events.

COSMO defines several properties to describe the temporal characteristics of entities. In this work only the properties used to support the PA and the IA will be presented. The properties cosmo:occursBefore, cosmo:occursAfter and cosmo:equals are used to represent the $<$, $>$ and $=$ temporal relations respectively. The cosmo:occursBefore is defined as being inverse of cosmo:occursAfter and applicable to cosmo:TimePoint (by defining the property domain and range), and cosmo:equals is defined as being symmetric. In the Table 3.1 the mapping between thirteen Allen's IA relations and the corresponding COSMO properties is presented. The prefix of the properties is omitted due to document formatting purposes.

Despite defining the necessary PA and IA properties, OWL is not sufficiently expressive to represent and reason about them. To ensure the algebras reasoning capability several SWRL rules were added to the ontology. The algebra is applicable to every instance of TemporalThing. Since intervals are represented by stating both starting and ending points, the interval algebra was defined according to the correlation between Allen's IA relations and Vilain PA relations presented in section 2.1.3. The rule presented by the

Relation Name	**COSMO Property**	**Inverse COSMO Property**	**OWL Property Characteristics**
equals	equals		
before	occurredEarlierThan	occurredLaterThan	Transitivity
meets	meetsTemporally	metByTemporally	
overlaps	overlapsTemporally	overlapedByTemporally	
during	during	containsTemporally	Transitivity
starts	starts	startedBy	
finishes	finishes	finishedBy	

Table 3.1: Thirteen Allen's Interval Algebra relations and corresponding COSMO Properties

SWRL rule 3.1 ensure the overlaps relation.

$$\begin{aligned}&\text{TemporalThing}(?tt1) \wedge \text{TemporalThing}(?tt2) \wedge \text{TimePoint}(?etptt1) \wedge \\ &\text{TimePoint}(?etptt2) \wedge \text{TimePoint}(?stptt1) \wedge \text{TimePoint}(?stptt2) \wedge \\ &\text{hasEndingTimePoint}(?tt1,\ ?etptt1) \wedge \text{hasEndingTimePoint}(?tt2,\ ?etptt2) \wedge \\ &\text{hasStartingTimePoint}(?tt1,\ ?stptt1) \wedge \text{hasStartingTimePoint}(?tt2,\ ?stptt2) \\ &\wedge \text{occursBefore}(?etptt1,\ ?etptt2) \wedge \text{occursBefore}(?stptt1,\ ?stptt2) \wedge \\ &\text{occursBefore}(?stptt2,\ ?etptt1) \\ &\qquad \rightarrow \text{overlapsTemporally}(?tt1,\ ?tt2)\end{aligned}$$

SWRL Rule 3.1: Modeling overlaps temporal relation using using SWRL

The necessary SWRL rules to model the rest of the PA relations were implemented analogously.

There are also some properties with more particular temporal semantics. The property cosmo:isaTemporalPartOfEvent is used to relate a cosmo:Event to another one which it is a part of. This property has cosmo:hasEventTemporalPart as its inverse and it is a transitive property. The property cosmo:hasRoleFiller (which inverse property is cosmo:fillsTheRoleOf) relates an instance of cosmo:Role with the entity that fills that role.

The spatial concepts are represented by subclasses of cosmo:SpatialThing, which represents the class of things that have spatial extent or location relative to other spatial thing or some embedding universe. According to COSMO, a spatial thing can be something tangible (*e.g.*, a physical object), partially tangible (*e.g.*, a country) or intangible (*e.g.*, a line mentioned on a geometric theorem). It does not make any assumption regarding on which universe the spatial thing is located; it could or not be on the actual physical universe. The two main subclasses of cosmo:SpatialThing are cosmo:SpatialThing-Localized and cosmo:Region. The class cosmo:SpatialThing-Localized allows to represent all tangible

Relation Name	COSMO Property	Inverse COSMO Property	OWL Property Characteristics
DC	isSpatiallyDisjointWith		Symmetry
EC	externallyConnected		Symmetry
PO	partiallyOverlapping		Symmetry
TPP	isTangentialProper-PartOf	hasTangentialProper-Part	
NTPP	surroundsCompletely	isCompletelySurround-edBy	
EQ	equals		Symmetry
NTPPi	isCompletelySurround-edBy	surroundsCompletely	

Table 3.2: The RCC8 relations and corresponding COSMO Properties

or intangible spatial things that have location or position in the empirically observable universe in question and that are temporal things. These include cosmo:GeographicalThing (entities that are localized within the context of some geography and thus can be represented on a map) and cosmo:Situation-Localized (the category of things that happen and whose temporal extent occurs at some specific location in space, which itself includes cosmo:Event).

The class cosmo:Region includes both basic entities of space, namely extended entities (cosmo:SpatialRegion) and points (cosmo:SpacePoint). Analogously to the temporal representation, cosmo:SpacePoint is subclass of cosmo:SpatialRegion since they assumed to be represented as regions of zero dimensions. Other specializations of cosmo:SpatialRegion is the class cosmo:PhysicalSpaceRegion which is a portion of the n-dimensional space of human real space-time universe and also a subclass of cosmo:SpatialThing-Localized. Instances of cosmo:SpatialRegion are usually assumed to be stationary for practical purposes. For non-stationary regions cosmo:SpatiotemporalRegion should be used.

Similarly to time, COSMO also defines several properties to describe the spatial characteristics of entities. Although COSMO includes properties which allow defining some of the RCC8 relations, namely the DC (cosmo:isSpatiallyDisjointWith), NTPP (cosmo:surroundsCompletely), NTPPi(cosmo:isCompletelySurroundedBy) and EQ (cosmo:equals) relations, there are not any properties whose semantics matches the EC, PO, TPP and TPPi relations. For this purpose four new properties have been defined, namely cosmo:externallyConnected, cosmo:partiallyOverlapping and cosmo:isTangentialProperPartOf, cosmo:hasTangentialProperPart. In the Table 3.2 is pre-

sented the mapping between the RCC8 relations and the corresponding COSMO properties.

3.2 Case Study: Football Ontology

In order to support the explanation of the engineering of temporal and spatial knowledge in domain ontologies will be used an ontology about football.

Football Ontology is an ontology that model some aspects of the football domain and the relations between them, including the notion of Team, Stadium, Match, Competition, Season and several possible roles represented by persons, namely being the manager of a club or a football player. Although every ontological concept (class, property or individual) is identified by a self-explanatory name following the camel case notation[2], an additional description will be provided. Figure 3.2 depicts the class hierarchy of the Football Ontology.

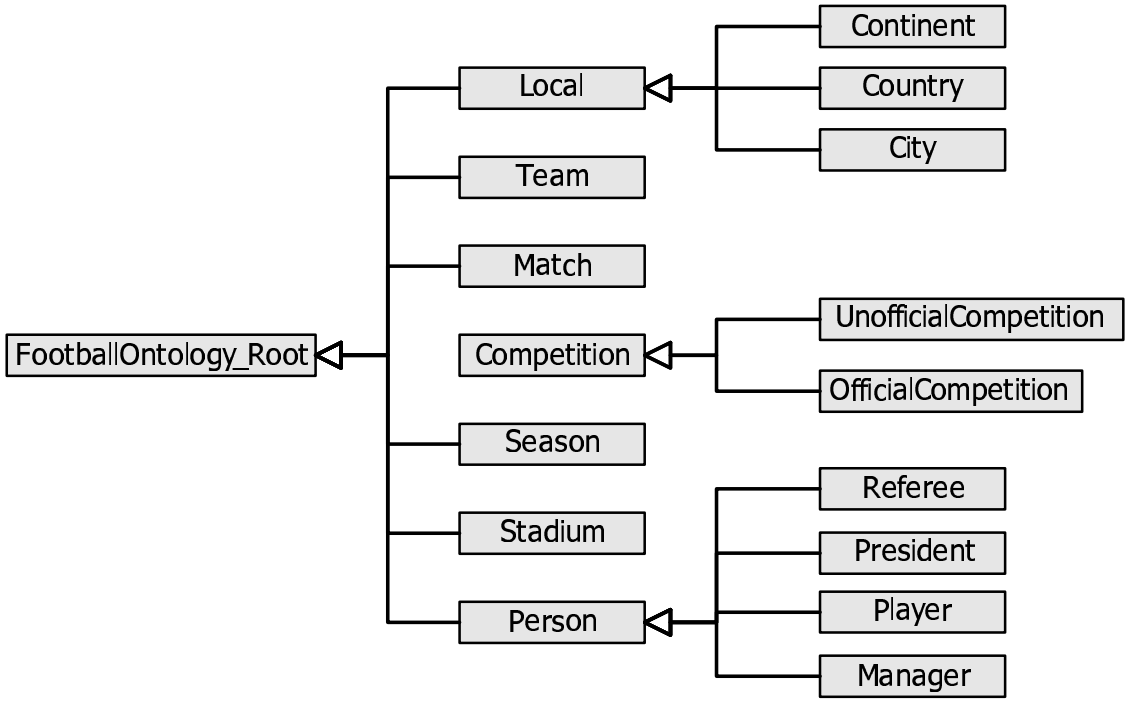


Figure 3.2: Class Hierarchy of Football Ontology

foot:Team – represents the set of the football teams. Each team must have a manager, a stadium and is the team of some football player. Examples of team individuals include foot:ManchesterUtd, foot:RealMadridCF and foot:SportingCP;

[2] `http://en.wikipedia.org/wiki/CamelCase`

foot:Stadium – represents the set of the football stadiums, the physical infrastructure where matches are played. Each stadium must be the stadium of some team and be the stadium of some match;

foot:Match – represents the set of football matches between two teams. Every match must have several referees (depending of the match rules, or the competition rules if the match is included in some), be played at some Stadium and have two opponent teams, namely the visited team (also known as home team) and the visitor team. The concept of visited and visitor team is important in cases of two-legged tie matches, in which the goals scored by the visitor can be used to untie the match and establish the winner;

foot:Competition – represents the football competitions, including official national (*e.g.*, Premier League - the English main football league), and international (*e.g.*, Champions League - the most important international league of clubs, played mainly in Europe, which include the winners of several national European leagues) or unofficial, which usually consists of one or several friendly matches;

foot:Season – represents a competition played during a specific period. For instance, the 2013/14 Premier League Season is the 22nd season of the Premier League, the English professional league for association football clubs, since its establishment in 1992. A Season must have a winner;

foot:Person – represents the persons that may be involved in all previous concepts. The classes foot:Player, foot:Manager, foot:Referee and foot:President are all specializations of the class foot:Person. A player is a player of some team; a manager is a manager of some team; a referee is the referee of some match; and a president is the president of some team.

The purpose of the Football Ontology is to represent and reason about the evolution of the football knowledge. However, it lacks of formal temporal and spatial representation. Without the ability to represent the time it is not possible to model some FIFA rules related to Status and Transfer of Players, namely:

- Chapter III: Registration of Players, Article 5.2: A player may only be registered with **one club at a time**.
- Chapter III: Registration of Players, Article 5.3: Players may be registered with a

maximum of three clubs during one season. During this period, the player is only eligible to play official matches for two clubs. (. . .)

- Chapter III: Registration of Players, Article 5.4: Under all circumstances, due consideration must be given to the sporting integrity of the competition. In particular, a player may not play official matches for more than two clubs competing in the same national championship or cup during the same season, subject to stricter individual competition regulations of member associations.

Similarly, it is not possible to model other domain temporal restrictions such: "A person can not be simultaneously football player and/or manager and referee".

3.3 Engineering Time and Space in OWL Domain Ontologies

In this section is shown how temporal and spatial knowledge can be engineered in OWL domain ontologies following the endurantist approach. This approach consists of applying changes to both ontology TBox and ABox. Performing the temporal and spatial engineering over the ontology TBox increases the reasoning capabilities by providing formal definitions for classes, properties and restrictions. The temporal and spatial engineering process involves structural changes over the ontology TBox, which have direct impact on how ABox entities must be represented.

In this section is shown how to temporal engineer ontology classes, properties and restrictions. It is also shown the impact that it has in the ontology individuals and their relations and to properly model them.

3.3.1 Temporal and Spatial Engineering of Classes

Defining temporal and spatial domain classes is usually done by adding restrictions specifying the beginning and the end of the temporal validity and the spatial location, respectively. In this work, temporal and spatial aspects are added by relating the domain class to one of several temporal and spatial classes. In COSMO, the two more generic classes of time and space are cosmo:TemporalThing and cosmo:SpatialThing. The temporal and/or spatial engineering of classes is done by creating a subclass relation between the target domain class and the COSMO temporal and/or spatial class. By doing this, the domain class inherits the temporal/spatial characteristics of the superclass, namely the existence of the temporal validity and spatial location and the class semantics that may be explored

by some system. For instance, the domain class foot:Season may be considered temporal, since every football season has a beginning and ending time point. To model this situation a subclass relation is added between foot:Season and cosmo:TemporalThing as depicted in Figure 3.3.

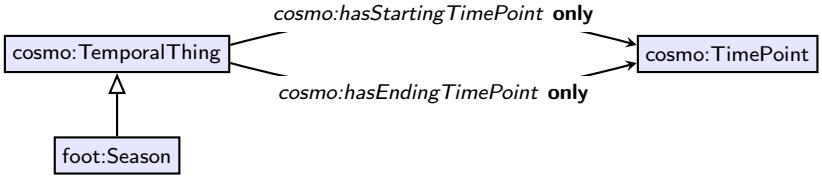


Figure 3.3: Temporal Engineering of the class cosmo:Season

Similarly, the class foot:Stadium may be considered a spatial concept, since every Stadium has a spatial location, and so a subclass relation between foot:Stadium and cosmo:SpatialThing is added as depicted in Figure 3.4.

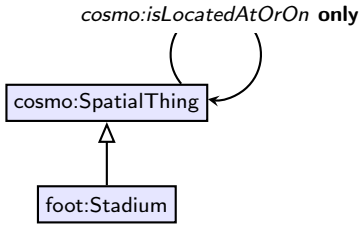


Figure 3.4: Spatial Engineering of the class cosmo:Stadium

It is also possible to use more specific temporal/spatial concepts, such events cosmo:Event, cosmo:GeographicalThing or cosmo:Role. By using more specific classes in the temporal/spatial engineering is considered a more restricted semantics. For instance, the class foot:Match may be considered an event, since it denotes something that happens at a certain moment and in a particular location, namely in a Stadium (see Figure 3.5). By its turn, a stadium can be considered a geographical thing, since it consists in an entity which is located within a geographical context, in the sense that it can be represented on a map.

It should be noted that the ontology engineer can decide to use even more specific types. Such a decision must be made according to user needs.

The engineering of temporal classes using temporal roles requires particular attention. The difference between saying that a footballer is a person or that footballer is a role played by a person may seem purely conceptual. However, it has a huge impact on how the system may be modeled and how the model should be reasoned and queried.

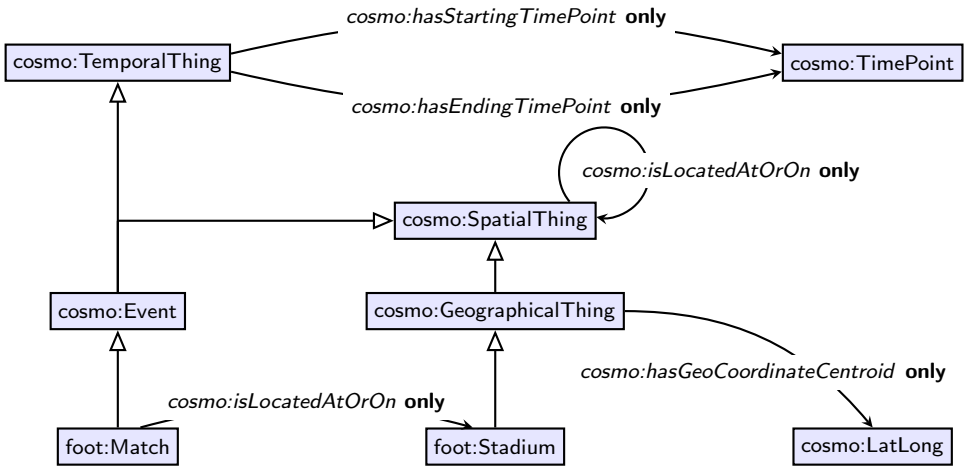


Figure 3.5: Temporal and Spatial Engineering of the class cosmo:Match

Typically there are entities which fit a particular type only for a period of time. For intance, consider the case of the evolution of the bee. Initially a bee starts as an egg and as it grows it passes through different stages, including larvae, pupae and adult bee. Likewise, humans can play several roles throughout their life and some of them simultaneously. Also inanimate objects can be classified in different ways according to the purpose for which they are used. A small table can be used as a chair if that is their purpose. Regarding the football domain a person can perform various functions, such as football player, manager or referee. Some of them may occur simultaneously (football player and manager) while others may only occur in different periods (football player and referee). It seems likely that the classification of entities do not necessarily have to be rigid, in the sense of being atemporal. In [Guarino and Welty, 2000] Guarino and Welty describe a formal ontology of properties and present the basic types of properties. These types include the notion of role, distinguishing them in formal and material roles. In OWL there are no formal way to represent roles. All classifications are assumed to be rigid, lasting throughout the life of the entity.

The two most common ways to assign roles to entities are the simple assignment of a type or the creation of an auxiliary entity that represents the role played by the first and also the corresponding relation between them. In the first case it is not made a distinction between types and roles, considering only types. Thus, to represent the fact that a person is a football player the entity is simply classified has instance of the class foot:FootballPlayer. Although this may be the most intuitive way of modeling the representation of roles it raises some problems. In OWL there is no formal way to represent the temporal classifica-

tion of instances since, like all relations, the type relation is also binary. To represent the temporal argument the type relation must become ternary and this is not straightforward as it will be discussed in section 3.3.2. Although possible, the typeOf relation (one of the most important relations in OWL) becomes too complex, compromising the ontology reasoning capabilities. This prevents the correct modeling of some domain rules, namely "A person can not be simultaneously football player and/or manager and referee".

Another way to model the representation of roles is to explicitly define which classes correspond to roles by creating a subclass relation between the domain class and the class cosmo:Role. Subsequently, for each individual who plays a role it should be created a new instance of the type of the played role and established the respective relation between the individual and the role he plays.

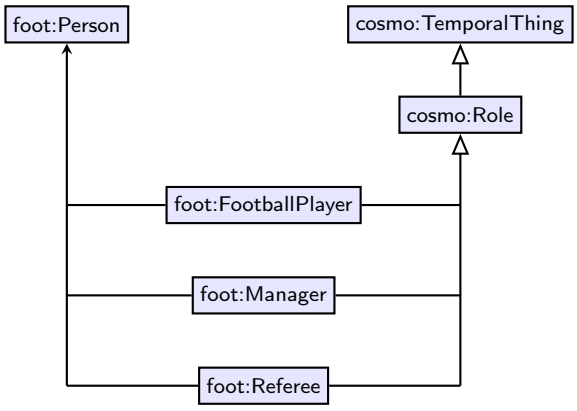


Figure 3.6: Temporal Engineering of several classes as Temporal Roles

In the football ontology the three subclasses of cosmo:Person can be considered roles played by a person. Hence, a subclass relation between cosmo:Role and foot:FootballPlayer, foot:Manager and foot:Referee may be created, and also a cosmo:hasRoleFiller relation between each one of them and foot:Person (see Figure 3.6).

Every instance of each one of the roles may also incur in changes. Consider the existence of an individual foot:alexFerguson that represents Sir Alex Ferguson, former football player and manager of several teams. This individual is related to foot:ManchesterUtd through the property foot:managerOf and to foot:Rangers through foot:hasTeam (see Figure 3.7).

The instance foot:alexFerguson, previously classified as foot:FootballPlayer and foot:Manager may now be classified only as foot:Person (see Figure 3.8).

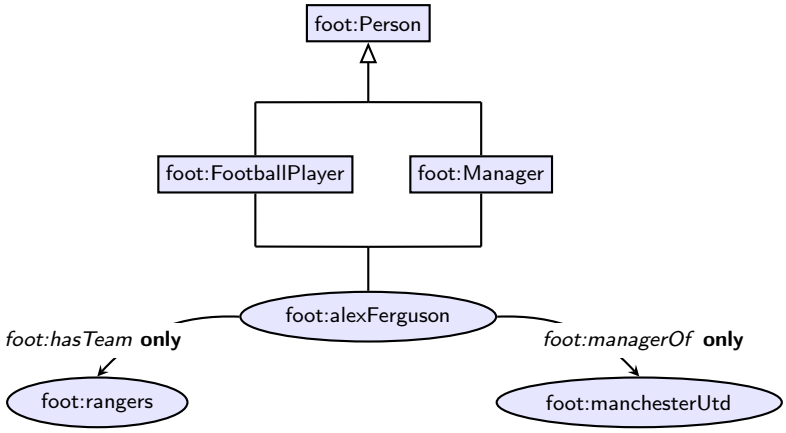


Figure 3.7: Instance representing Alex Ferguson and its relations before the temporal engineering

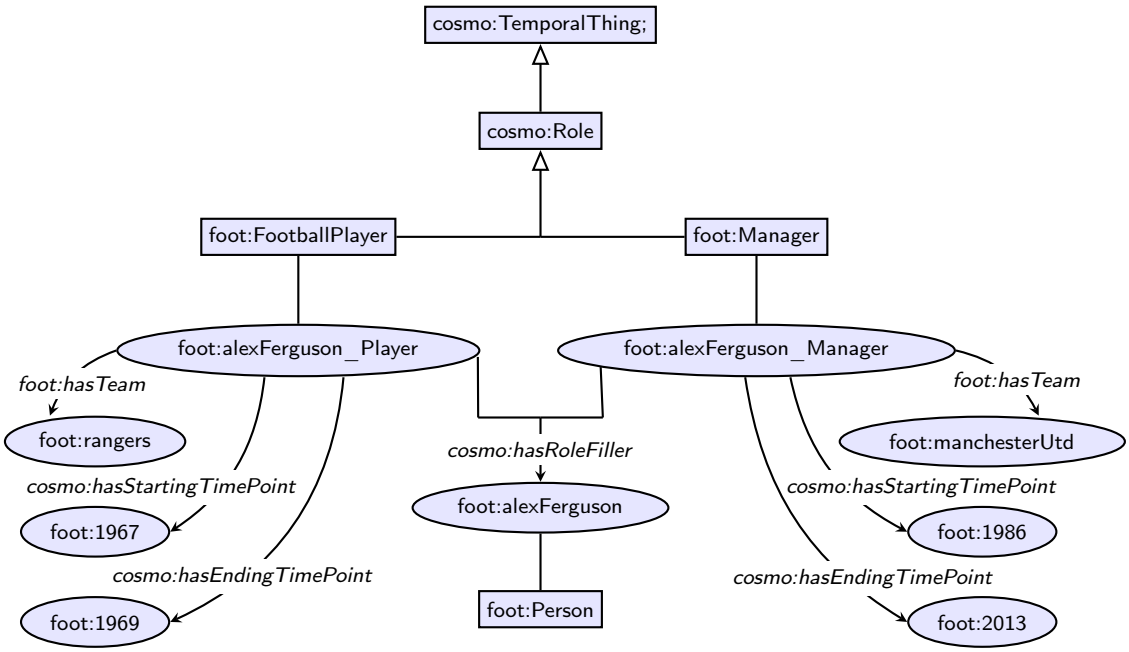


Figure 3.8: Instance representing Alex Ferguson and its relations after the temporal engineering

To describe each one of the roles played by Alex Ferguson, two new individuals may be created, namely foot:alexFerguson_Player of type foot:FootballPlayer and foot:alexFerguson_Manager of type foot:Manager. Then, they must be related to the individual foot:alexFerguson through cosmo:hasRoleFiller property. Both foot:managerOf and foot:hasTeam relation must be removed from the instance foot:alexFerguson and be added to foot:alexFerguson_Manager and foot:alexFerguson_Player respectively, since they concern the role played by the entity.

It must be noted that, after performing all changes needed to explicitly model roles, questions such "Which teams were managed by Alex Ferguson?" becomes necessarily

more complex to realize, since it is necessary to search the relations foot:fillsTheRoleOf from the instance foot:alexFerguson, go through the role of type foot:Manager played by him and only then find the teams through the foot:managerOf relation.

Specifying simultaneously incompatible roles is also much more difficult, such playing the role of football player and referee at the same time. Before considering temporal classification or temporal roles it was only necessary to specify that the two classes are disjoint. Now it is necessary to ensure that no individual of foot:Person can play any two roles considered disjoint during the same period of time. This is not possible to specify using only OWL axioms. Instead, a SWRL rule must be created for each role playing restriction. The SWRL 3.2 correspnds to the rule that ensures the temporal incompatibility between playing the role of football player and referee at the same period of time.

foot:Person(?p) ∧ foot:Referee(?r) ∧ foot:FootballPlayer(?fp) ∧
cosmo:fillsTheRoleOf(?p,?r) ∧ cosmo:fillsTheRoleOf(?p,?fp) ∧
cosmo:overlapsTemporally(?r,?fp)

→ owl:Nothing(?p)

SWRL Rule 3.2: Detecting temporal inconsistency in instance role playing

3.3.2 Temporal and Spatial Engineering of Restrictions

The temporal/spatial engineering of restrictions is much more complex. For instance, consider the team relation between a football player and a football team. Each football player can play for many teams during his/her career and so each instance of foot:FootballPlayer can have multiple team relations with each foot:FootballTeam (see Figure 3.9).

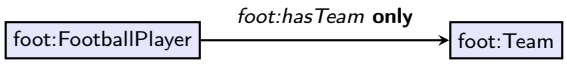


Figure 3.9: Restriction stating that the team of a football player can be only of type foot:Team

This is not problematic in OWL, since each instance can have multiple relations to other instances, unless explicitly specified. However, it is not possible to easily specify the temporal validity of each relation, as is necessary to do according to the endurantist approach. Since OWL is restricted to binary relations it is not possible to simply add the temporal/spatial argument in the relation (see Figure 3.10).

Without the ability to represent the temporal validity of each relation it is not possible to

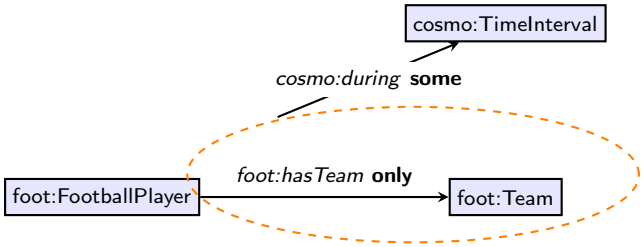


Figure 3.10: Problem of adding a third argument when using binary based languages

model some FIFA rules related to Status and Transfer of Players, namely those presented in section 3.2 concerning the Status and Transfer of Players.

The most common approach to deal with predicates of higher arity than two in languages based in binary relations is to introduce a new concept to play the role of n-ary relation. In OWL, this is usually done by following the N-ary relations ODP. This ODP is well documented [Noy and Rector, 2006] and will be used in the use case example. According to this ODP, a new class must be introduced to play the role of N-ary relation. The new class is then linked to the subject of the relation and to the several objects through each property. For instance, in order to temporalize the restriction foot:FootballPlayer foot:hasTeam only foot:Team is necessary to create a new class (*e.g.*, foot:NAryTeamRel), add a foot:hasTeam relation between the original subject foot:FootballPlayer and the new class and add other foot:hasTeam relation between the new class and the original subject foot:Team (see Figure 3.11). The new class is made subclass of cosmo:TemporalRelation which in turn is itself subclass of cosmo:TemporalThing and so inherit the obligation to have a temporal beginning and end. The temporal beginning and end of the new auxiliary class corresponds to the temporal validity of the relation.

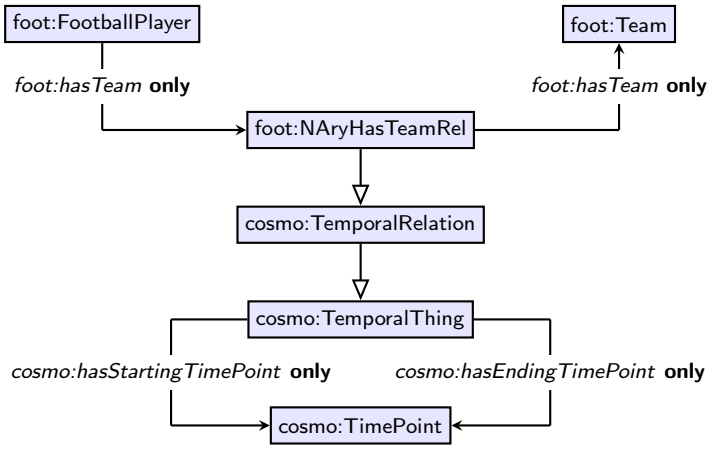


Figure 3.11: Modeling N-ary relation according to the N-ary relations pattern

The instances of foot:FootballPlayer also have to incur in changes. Consider the instance foot:CristianoRonaldo, which represents the football player Cristiano Ronaldo that played for Sporting CP, Manchester United and currently plays for Real Madrid. Therefore, the foot:CristianoRonaldo instance has several foot:hasTeam relations, each one related to one of the clubs that Cristiano Ronaldo represented (see Figure 3.12).

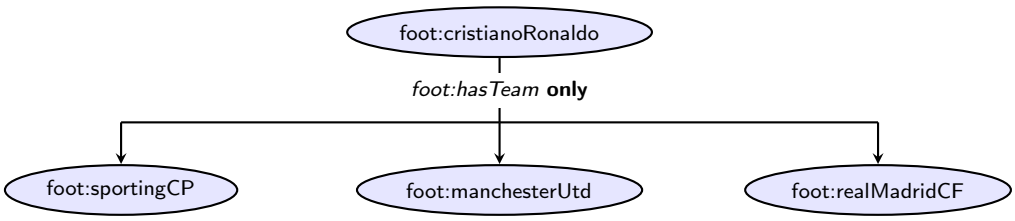


Figure 3.12: Relation between Cristiano Ronaldo and the several teams in which he played before the temporal engineering

Since now the foot:hasTeam relations are considered temporal, they must incur in changes similarly to what have been made at class level. Hence, three new instances of foot:NaryTeamRel are introduced to play the role of N-ary relation, namely foot:CR_team_SportingCP, foot:CR_team_ManchesterUtd and foot:CR_team_RealMadridCF, each one related with the corresponding foot:Team and with the beginning and end time points (see Figure 3.13).

It should be note that the temporal validity of each relation is knowledge that was not previously available, so it must be added *a posteriori*.

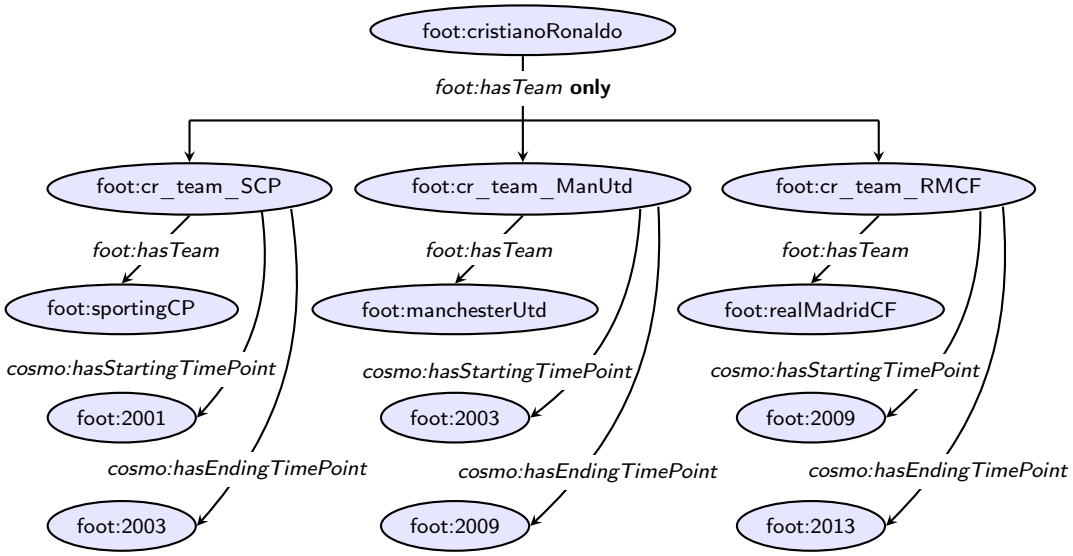


Figure 3.13: Relation between Cristiano Ronaldo and the several teams in which he played after the temporal engineering

It is still needed to perform some changes at class level, in order to ensure the ontology consistency and correct inferences. The property foot:hasTeam has the inverse property foot:teamOf. Moreover, a restriction that states that every player of a foot:Team must be an instance of foot:FootballPlayer is also defined in the ontology, as depicted in Figure 3.14.

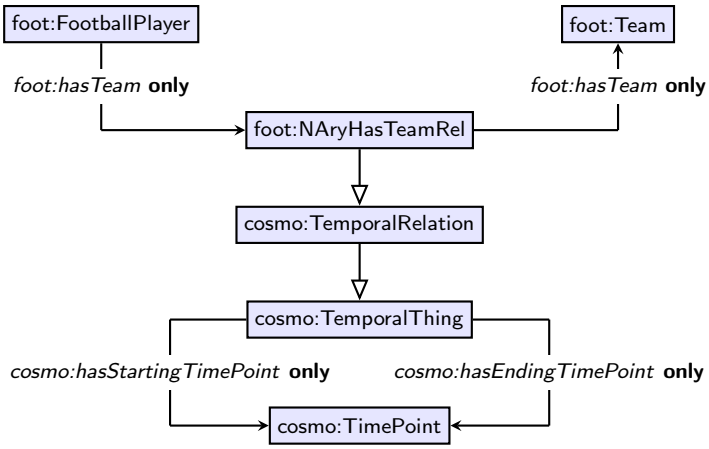


Figure 3.14: Inverse restriction on properties foot:hasTeam/foot:teamOf

In order to allow the inference of the inverse relation, the foot:teamOf inverse restriction has also to incur in changes. These modifications occur similarly to those performed on the foot:hasTeam restriction, but in this case no additional class needs to be created, since the previously created class will be reused (see Figure 3.15).

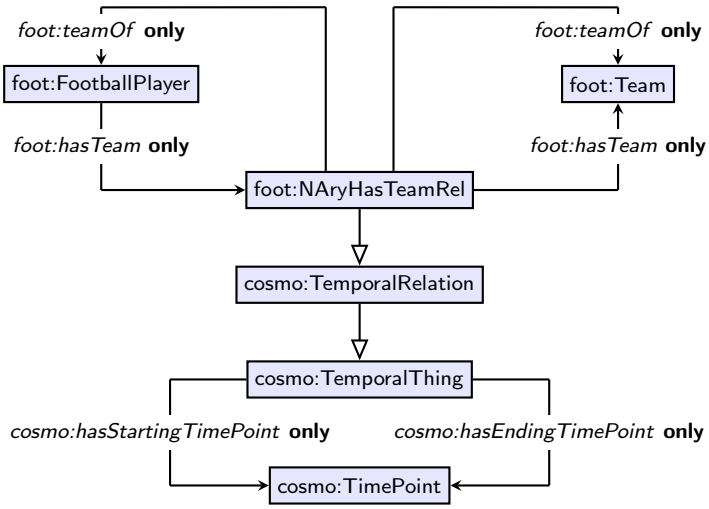


Figure 3.15: Ensuring inverse restriction on properties foot:hasTeam/foot:teamOf after temporal engineering

If the properties used in the previously modified relations specify domain and range, some additional changes needed to be made. For instance, consider that the property

foot:hasTeam defined that its domain is foot:FootballPlayer. By doing that, every instance that acts as subject in a foot:hasTeam relation should be classified as foot:FootballPlayer. Due to the necessary changes performed to make the restriction temporal, the system will now be able to conclude that foot:CR_team_SportingCP, like all other instances created to play the role of the N-ary relation, are instances of the class foot:FootballPlayer. This conclusion is logically valid, but semantically wrong. In order to correct this situation, the domain of the foot:hasTeam property can not be the class foot:FootballPlayer but, instead, it must be the union of foot:FootballPlayer and foot:NaryTeamRel. The same changes need to be performed to the property range.

3.3.3 Temporal and Spatial Engineering of Properties

When considering that a property is temporal, in the sense that it is used in relations that can change through time, some considerations about the property characteristics have to be made, in particular concerning transitivity, functionality, symmetry, reflexivity and inverse property.

3.3.3.1 Transitive Properties

Set a property as transitive means that if the property relates the individual x with the individual y and also the individual y with the individual z then it can be concluded that the individual x must be also related with the individual z through that property (see Axiom 3.3.1).

$$\forall x, y, z : P(x, y) \wedge P(y, z) \rightarrow P(x, z) \tag{3.3.1}$$

For instance, consider the transitive property foot:hasGreaterCapacityThan which can be used to relate two instances of foot:Stadium to define that a stadium has greater capacity than another one. In a dynamic context this characteristic could change, since sometimes the clubs carry out changes in their stadiums that can cause a decrease or increase of the maximum capacity. Consequently the transitivity of the foot:hasGreaterCapacityThan has to consider the temporal validity of the defined relations. Consider the following three stadiums:

Highbury – the former stadium of Arsenal Football Club;

Stamford Bridge – the actual stadium of Chelsea Football Club;

Anfield Road – the actual stadium of Liverpool Football Club.

These stadiums are respectively represented by the instances foot:HighburyStadium, foot:StamfordBridge and foot:AnfieldRoad.

The maximum seating capacity of each stadium as suffered some changes during their life time. For instance, consider that someone defined that the Anfield Road had a greater seating capacity than Stamford Bridge from year 1950 to year 1980 (see Figure 3.16). It was also defined that Stamford Bridge had a greater seating capacity than Highbury from year 1960 to year 1990 (see Figure 3.17).

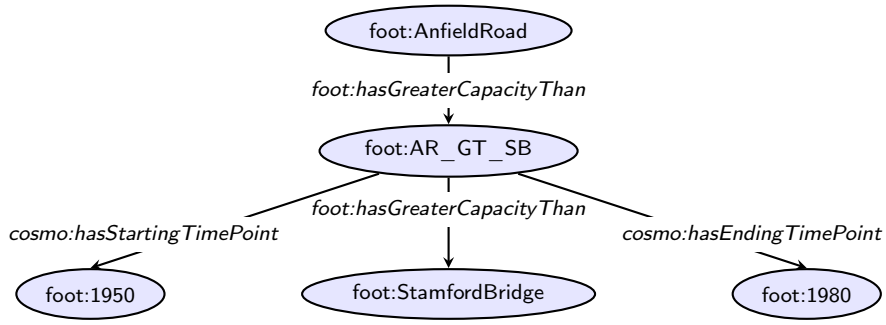


Figure 3.16: Temporal hasGreaterCapacityRelation between foot:AnfieldRoad and foot:StamfordBrideg

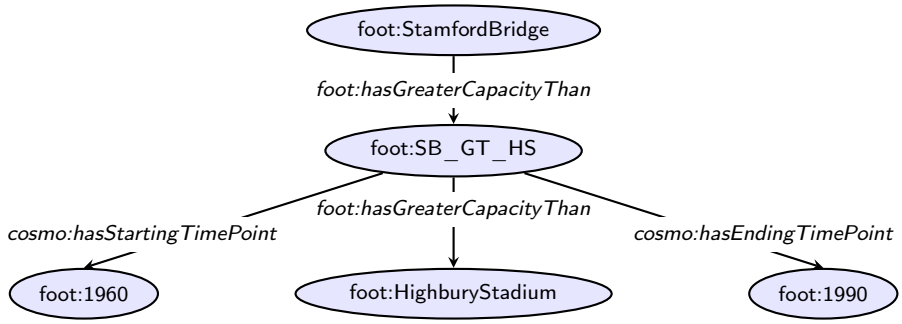


Figure 3.17: Temporal hasGreaterCapacityRelation between foot:StamfordBridge and foot:HighburyStadium

By exploring the transitivity of the foot:hasGreaterCapacityThan and considering that the relations are temporal is possible to conclude that Anfield Road had greater seating capacity than Highbury stadium from year 1960 to year 1980. Before and after that period nothing can be inferred, since there is no sufficient information available (see Figure 3.18).

Using only OWL axioms it is not possible to ensure the temporal transitivity, but it can be done at instance level by using SWRL rule. The SWRL rule 3.3 is used to ensure the

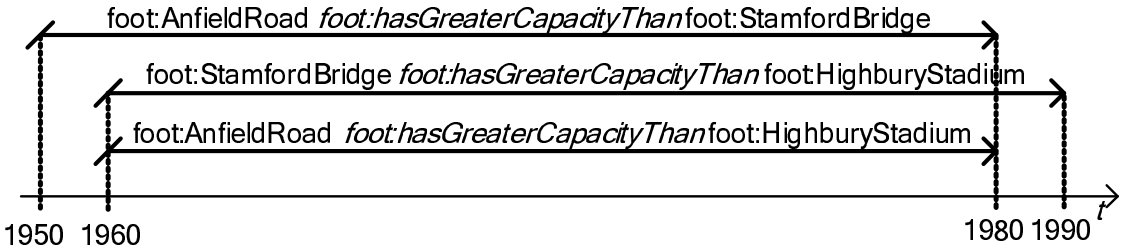


Figure 3.18: Example of temporal transitivity of the property foot:hasGreaterCapacityThan

temporal transitivity of the foot:hasGreaterCapacityThan property.

cosmo:TemporalRelation(?tr1) ∧ football(?tr2) ∧
foot:hasGreaterCapacityThan(?x,?tr1) ∧
cosmo:hasStartingTimePoint(?tr1,?tr1_stp) ∧
cosmo:hasStartingTimePoint(?tr2,?tr2_stp) ∧
cosmo:greaterThan(?tr2_stp,?tr1_stp) ∧
foot:hasGreaterCapacityThan(?tr1,?y) ∧
foot:hasGreaterCapacityThan(?y,?tr2) ∧
foot:hasGreaterCapacityThan(?tr2,?z) ∧
cosmo:overlapsTemporally(?tr1,?tr2) ∧
cosmo:hasEndingTimePoint(?tr1,?tr1_etp) ∧ swrlx:makeOWLThing(?tr3)
→ cosmo:TemporalRelation(?tr3) ∧
foot:hasGreaterCapacityThan(?x,?tr3) ∧
foot:hasGreaterCapacityThan(?tr3,?z) ∧
cosmo:hasStartingTimePoint(?tr3,?tr2_stp) ∧
cosmo:hasEndingTimePoint(?tr3,?tr1_etp)

SWRL Rule 3.3: Ensuring temporal transitivity of foot:hasGreaterCapacityThan property

However, the SWRL rule 3.3 makes use of the swrlx:makeOWLThing experimental SWRL built-in that creates OWL individuals at runtime, which is not supported by the majority of the reasoners for safety reasons as discussed in section 2.2.2. Since to model temporal relations it is necessary to introduce a new entity to play the role of the temporal relation (as explained in section 3.3.2), the SWRL rule may have the ability to create new individuals at runtime. Despite there is no expedit way to do that it is possible to overcome this problem by using the commercial reasoner Jess Back-End.

3.3.3.2 Functional Properties

Set a property as functional means that it can not be used in more than one relation between a particular subject individual and different object individuals (see Axiom 3.3.2). If the individuals are not explicitly different, then an inference mechanism must consider

that they are the same (see Axiom 3.3.3).

$$\forall x, y : P(x, y) \wedge P(x, z) \wedge owl : DifferentFrom(y, z) \rightarrow \perp \tag{3.3.2}$$

$$\forall x, y : P(x, y) \wedge P(x, z) \rightarrow owl : SameAs(y, z) \tag{3.3.3}$$

This can be seen as a special case of max cardinality restriction. When considering dynamic ontologies the functionality can be interpreted in two ways:

Absolute functionality – when the purpose is to impose at most one relation between a subject individual and any different object individuals regardless of the temporal validity of the relation. For instance, consider that any person can have at most one official FIFA Registry Number, and that that number is permanent. In that case the property foot:officialFIFARegistryNumber must be absolutely functional. In OWL, the functional property axiom can be used to model cases of absolute functionality.

Temporal functionality – when the purpose is to impose at most one relation between a subject individual and any different object individuals only if the relations temporally overlap at some point in time. For instance, consider the foot:hasTeam property that can be used to relate a footballer to the club for which he plays. The article III-5.2 of the FIFA rule about Status and Transfer of Players states that "A player may only be registered with one club at a time". Using only OWL axioms it is not possible to conveniently model the temporal functionality. However, it can be assured at the instance level by implementing SWRL rules. The SWRL rule 3.4 specifies that for every relation through the property foot:hasTeam, if the subject is the same, the property is temporally functional and there are at least two different object individuals, then the ontology is inconsistent.

foot:hasTeam(?s,?o1) ∧ foot:hasTeam(?s,?o2) ∧ owl:DifferentFrom(?o1,?o2) ∧
cosmo:overlapsTemporally(?o1,?o2)
→ owl:Nothing(?s)

SWRL Rule 3.4: Ensuring temporal functionality of foot:hasTeam property

According to the functional property semantics if it is not explicitly specified that the relation objects are different, and assuming that there is some temporal overlap between

the temporal relations, the system may conclude that the individuals acting as object of the relations are the same. The SWRL rule 3.5 allows that kind of inferences.

Football:hasTeam(?s,?o1) ∧ foot:hasTeam(?s,?o2) ∧
foot:hasTeam(?o1,?team1) ∧ foot:hasTeam(?o2,?team2) ∧
cosmo:overlapsTemporally(?o1,?o2)
→ owl:SameAs(?team1,?team2)

SWRL Rule 3.5: Ensuring instance equivalence on temporal functionality of foot:hasTeam property

For instance, consider that someone states that Cristiano Ronaldo played for Manchester United from 2003 to 2006 (see Figure 3.19).

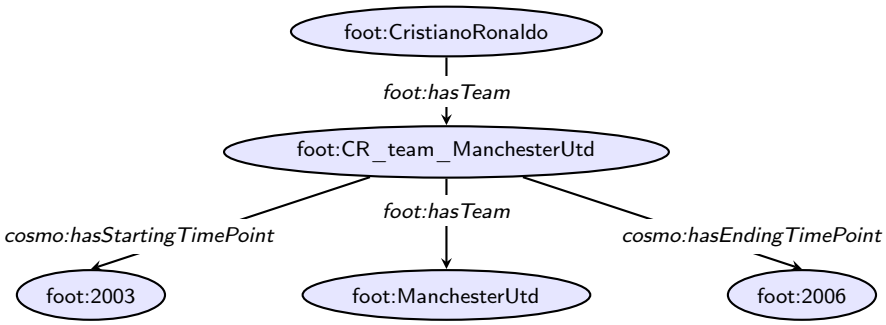


Figure 3.19: Temporal hasTeam between foot:CristianoRonaldo and foot:ManchesterUtd

Later, someone adds the fact that Cristiano Ronaldo played for a team represented by the instance foot:TheRedDevils from 2004 to 2008 (see Figure 3.20).

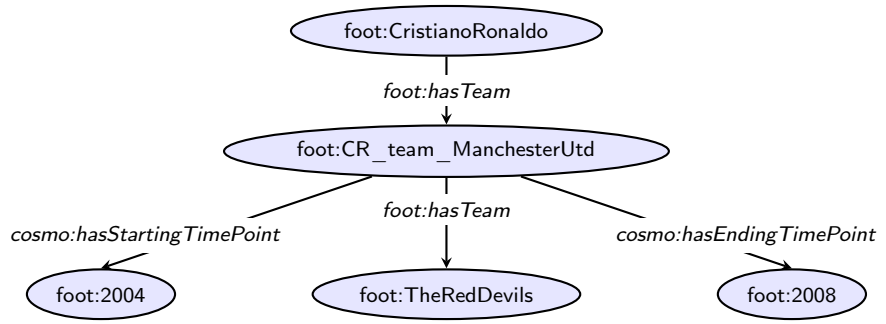


Figure 3.20: Temporal hasTeam between foot:CristianoRonaldo and foot:TheRedDevils

Since that foot:hasTeam has been defined as temporally functional, the instance foot:CristianoRonaldo has a foot:hasTeam temporal relation with more than one foot:Team and that those relations overlaps temporally, then the system must conclude that foot:ManchesterUtd and foot:TheRedDevils are two individuals that represent the same entity.

3.3.3.3 Inverse Properties

Specifying that two properties P1 and P2 are inverse of each other means that if an individual x is related to y through the property P1, then the relation on inverse direction through the property P2 must also exist (see Axiom 3.3.4).

$$\forall x, y : P1(x, y) \rightarrow P2(y, x) \tag{3.3.4}$$

For instance, consider the properties foot:hasTeam and foot:teamOf that are inverse of each other, and the relation foot:hasTeam between the instances foot:CristianoRonaldo and foot:ManchesterUtd. According to the semantics of the inverse property axiom an inference mechanism should conclude the relation foot:teamOf between the instances foot:ManchesterUtd and foot:CristianoRonaldo (see Figure 3.21).

Considering that the ontology becomes dynamic and the relation foot:hasTeam between a football player and a team is only valid during a period of time, then the inverse relation must also be valid during that period of time. As explained in section 3.3.2, temporal relations are modeled according to the N-ary relations ODP which implies the creation of a new entity to play the role of temporal relation.

No additional SWRL rules needs to be created in order to ensure the temporal inverse relations.

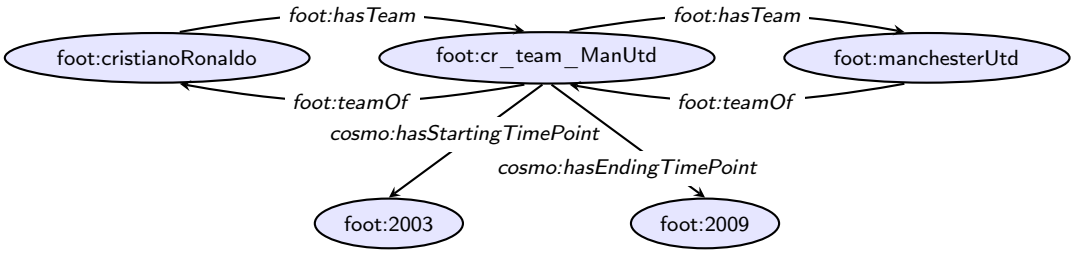


Figure 3.21: Ensuring inference of inverse relation after temporal engineering

3.3.3.4 SymmetricProperties

If a property P is then defined as being symmetrical, in the presence of a relation between two individuals x and y through P, a relation in the opposite direction and also through P must exist (see Axiom 3.3.5). This is a special case of the inverse relation, in which the property is opposite of her own.

$$\forall x, y : P(x, y) \rightarrow P(y, x) \tag{3.3.5}$$

For instance, consider the property foot:isPartnerOf that can be used to relate two teams that has some sort of agreement such the ability to loan out inexperienced young players, to allow young or foreign players to gain a work permit, or event for business purposes, such as merchandising. This property has been set as symmetric, since if the team A is partner of team B, then the team B is also partner of team A. In a dynamic context this relation can be considered temporal, since the partnerships between clubs often has a limited duration.

For example, consider the partnership relation between foot:SportingCP and foot:ManchesterUtd that started in the year 2003 and lasted till 2010.

Since the relation is temporal, the inverse relation inferred by exploring the property symmetry must also be valid only during the period of the first. However, unlike the case of the inverse property (see section 3.3.3.3), the correct inferred is not guaranteed by simply stating that the property is symmetric. In that case, the introduced entity would be used to specify the original relation and also the inferred relation (see Figure 3.22).

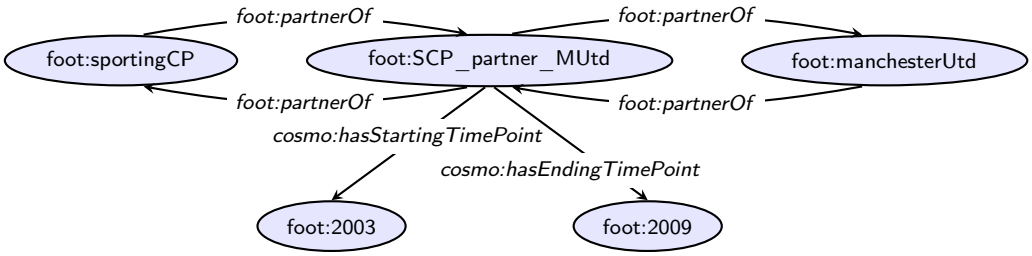


Figure 3.22: Inference results of inverse relation after temporal engineering

By doing this, two incorrect relations would be concluded, namely a temporal relation of partnership between each club and themselves (see Figure 3.22).

This is not what the system must infer since a football club is not partner of itself only during a period of time. The symmetry axiom must be replaced by the SWRL rule 3.6 that ensures the correct modeling of the temporal symmetry. The results are as depicted in Figure 3.23.

Like the SWRL rule 3.3 to ensure the temporal transitivity (see section 3.3.3.1) the SWRL rule 3.6 also needs to create a new instance at runtime, which can be problematic as

foot:Team(?t1) ∧ cosmo:TemporalRelation(?tr1) ∧ football(?tr2) ∧
foot:isPartnerOf(?t1,?tr1) ∧ foot:isPartnerOf(?tr1,?t2)
cosmo:hasStartingTimePoint(?tr1,?tr1_stp) ∧
cosmo:hasEndingTimePoint(?tr1,?tr1_etp) ∧ swrlx:makeOWLThing(?tr2)
→ cosmo:TemporalRelation(?tr2) ∧
foot:isPartnerOf(?t2,?tr2) ∧ foot:isPartnerOf(?tr2,?t1) ∧
cosmo:hasStartingTimePoint(?tr2,?tr1_stp) ∧
cosmo:hasEndingTimePoint(?tr2,?tr1_etp)

SWRL Rule 3.6: Ensuring temporal symmetry of foot:isPartnerOf property

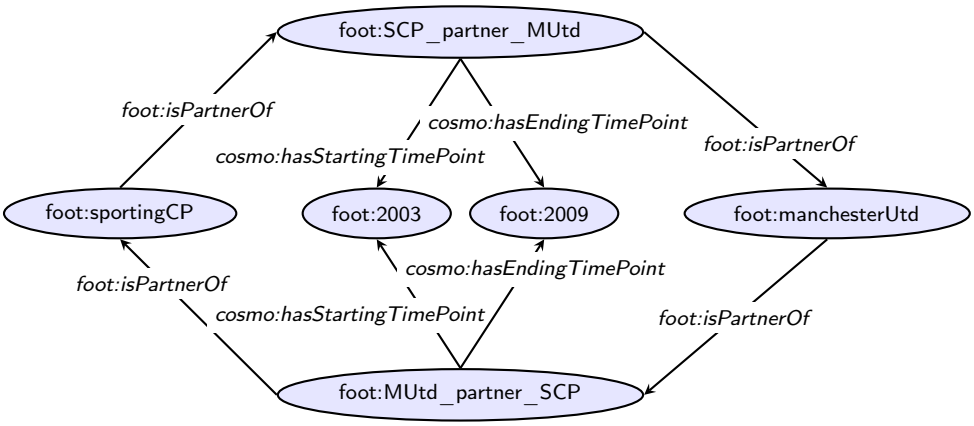


Figure 3.23: Correct modelling of temporal symmetry of foot:isPartnerOf property

discussed previously.

3.3.3.5 AsymmetricProperties

By defining a property P as asymmetric then if the individual x is related to individual y through the property P, y can not be connected to x through P (see Axiom 3.3.6).

$$\forall x, y : P(x, y) \rightarrow \neg P(y, x) \tag{3.3.6}$$

For instance, some football teams have a reserve team composed of players who need playing time, but do not actually have enough quality to play on the first team. The property foot:isReserveTeamOf, that can be used state that some team is the secondary team of other, is an asymmetric property. For example, the team Sporting C.P. owns a reserve team named Sporting C.P. B, which will be represented by the instance foot:SportingCP_B. However, a reserve team can become a primary team, like what happened with Atlético Malagueño, a former reserve team of CD Málaga which latter has been renamed to Málaga

CF[3].

So, the relation foot:isReserveTeamOf may be considered temporal. By doing that is necessary to ensure that the asymmetric characteristic of the property is also temporal. This means that if a team is reserve of another during some period of time, the latter can not be the reserve team of the former only during that period of time. This can be achieved by implementing the SWRL rule 3.7.

foot:Team(?t1) ∧ cosmo:TemporalRelation(?tr1) ∧ foot:Team(?t2) ∧
foot:isReserveTeamOf(?t1,?tr1) ∧ foot:isReserveTeamOf(?tr1,?t2) ∧
cosmo:TemporalRelation(?tr2) ∧ foot:isReserveTeamOf(?t2,?tr2) ∧
foot:isReserveTeamOf(?tr2,?t1) ∧ cosmo:overlapsTemporally(?tr1,?tr2)
→ Nothing(?tr1) ∧ Nothing(?tr2)

SWRL Rule 3.7: Ensuring temporal asymmetry of foot:isReserveTeamOf property

3.3.3.6 DisjointProperties

The property disjunction states that two disjoint properties can not be used to relate the same individuals at the same time. If the property P1 and P2 are disjoint then the individuals x and y can not be simultaneously related by P1 and P2 (see Axiom 3.3.7).

$$\forall x, y : P1(x, y) \wedge P2(x, y) \rightarrow \bot \tag{3.3.7}$$

For instance, consider the properties foot:isPartnerOf and foot:isRivalOf. It is reasonable to say that two football clubs can not be simultaneously partners and rivals, so the properties are defined as being disjoint of each other.

Considering that the ontology becomes dynamic and that the partnership and rivalry relations are considered temporal, then it is assumed that two instances can be related by both properties at different time periods. The SWRL rule 3.8 ensures the temporal disjunction of properties.

[3] `http://www.malagacf.com/en/club/history`

$$\begin{aligned}&\text{foot:Team}(?t1) \wedge \text{cosmo:TemporalRelation}(?tr1) \wedge \text{foot:Team}(?t2) \wedge \\ &\text{foot:isPartnerOf}(?t1,?tr1) \wedge \text{foot:isPartnerOf}(?tr1,?t2) \wedge \\ &\text{cosmo:TemporalRelation}(?tr2) \wedge \text{foot:isRivalOf}(?t1,?tr2) \wedge \\ &\text{foot:isRivalOf}(?tr2,?t2) \wedge \text{cosmo:overlapsTemporally}(?tr1,?tr2) \\ &\qquad\qquad \rightarrow \text{Nothing}(?tr1) \wedge \text{Nothing}(?tr2)\end{aligned}$$

SWRL Rule 3.8: Ensuring temporal disjointness between foot:isPartnerOf and foot:isRivalOf property

Chapter 4

FONTE – Factorize ONTology Engineering complexity

In this chapter is presented the re-engineered FONTE method and tool. In section 4.1 is presented the re-engineered FONTE method. In section 4.2 is presented the FONTE Assembly Process. The most important concepts of FONTE are described in the section 4.3, with particular attention to the Assembly notions of Task, Rule, Function and Proposal. Finally, in section 4.4 is presented the implementation of FONTE as a plug-in for Protégé, one of the most used ontology editors.

4.1 FONTE Method

The FONTE (Factorize ONTology Engineering complexity) method was re-engineered in order to support the engineering of temporal and spatial knowledge in OWL domain ontologies. FONTE uses a divide and conquer strategy by using ontologies as building blocks through the application of ODPs. The domain atemporal and aspatial ontology is developed first. Then, the temporal and spatial knowledge (represented in ontologies about time and space domain) is engineered step by step, through the so called Assembly Process. The final ontology consists in the domain ontology enriched with temporal and spatial notions (see Figure 4.1). FONTE promotes modularity and ontology reuse by allowing the separated development of each one of the ontologies and then their assembly. In section 3.3 was assumed a way of do it according to the endurantist approach for modeling time and space and by reusing ontologies about time and space, namely the COSMO ontology. This engineering process consists of performing several changes to the domain ontology entities. The descriptions of how these changes should be done correspond to the specification of ODP.

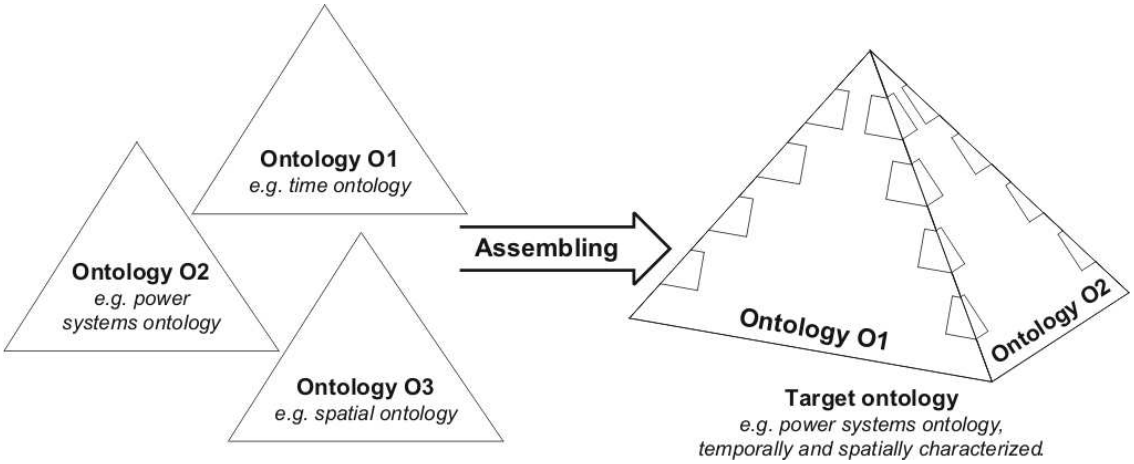


Figure 4.1: FONTE – Factorize ONTology Engineering complexity

ODPs are design solutions for recurring modeling problems. They consist of a set of directions that explain how to solve a problem under certain conditions. These directions are usually given in natural language and / or using graphic schemes. The use of ODPs is considered a best practice. However, their manual application may prove a difficult task since it may require the creation/removal/modification of several ontological entities which can make the ontology inconsistent and would require additional changes. Some software for creating / maintenance of ontologies support the automatic application of some of the most best known ODPs. However these patterns are hard coded in the software which make difficult to implement new patterns or change the existing ones.

Considering that there are several ways of engineering temporal and spatial knowledge in domain ontologies, namely by following other temporal and spatial theory (*e.g.*, perdurantist) or by reusing different temporal/spatial ontologies, the FONTE method was re-engineered to become flexible enough to support different options. The FONTE method allows defining ODP using Assembly Rules coded in a language developed for that purpose. Those rules are interpreted by FONTE allowing the automatic realization of large amounts of changes. The user only needs to indicate which Assembly Task (*e.g.*, to define a class, restriction or property as temporal or spatial) must be applied and all changes are consistently performed by FONTE.

In addition, FONTE acts as a rule based knowledge decision support system, by making suggestions for the applications of ODPs over some ontological entities, assisting the user in the choice of which entities must be changed and preventing him against possible forgetfulness.

During the Assembly Process the method is responsible to ensure the domain ontology consistency preventing the generation of conditions that generate inconsistencies such:

- circularity on subclass relations (*e.g.*, subClassOf(A,B) ∧ subClassOf(B,C) ∧ subClassOf(C,A))
- redundancy on subclass relations (*e.g.*, subClassOf(A,B) ∧ subClassOf(B,C) ∧ subClassOf(A,C))
- violation of OWL model semantics (*e.g.*, the creation of a subclass relation between two disjoint classes).

4.2 FONTE Assembly Process

The FONTE method implements an iterative and interactive assembly process. The assembly process comprises two main building blocks. First, the specification of temporal and/or spatial aspects for a domain ontology (atemporal and aspatial) remains dependent on the conceptualization of the ontology engineer. Second, in order to facilitate and accelerate the joint assembly of timeless and spaceless domain concepts with temporal and/or spatial notions, the interactive process is supported by heuristics for asking and directing the ontology engineer. The assembly process runs as depicted in Figure 4.2.

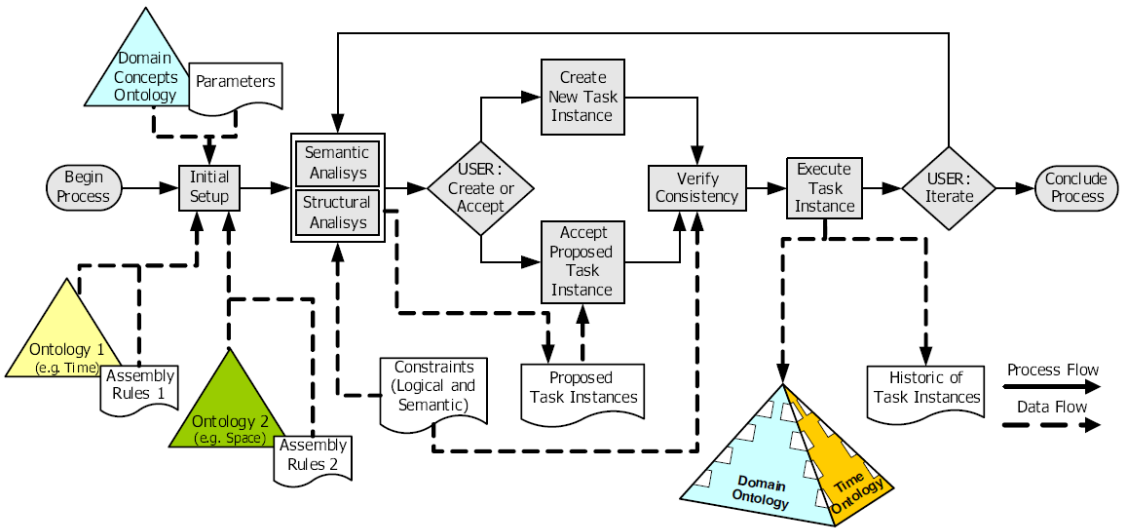


Figure 4.2: FONTE iterative and interactive assembly process

The process starts by an **Initial Setup**. Some basic operations are performed, namely loading the ontologies to be assembled, loading a set of Assembly Rules to drive the

process and initializing a set of process parameters. The Assembly Rules are defined separately from the tool in order to allow for adaptations to the particular needs of different transversal ontologies. However these rules do not change when new domain ontology is to be assembled since they only concern the transversal ontology and not the domain ontology. At this point the Target Ontology corresponds to the union of the domain and transversal ontology and consists of at least two separated clusters since the domain and transversal concepts are not yet related to each other.

The first step of the process consists in search potential candidate ontology entities to be temporally or spatially assembled by performing a **Semantic Analysis** over the participant ontologies and generating proposals for Assembly Task execution (which will going to be referred as Assembly Proposals). The strategies used at this step includes searching for ontology concepts that are usually considered temporal/spatial or that contains particular characteristics of temporal/spatial entities (*e.g.*, temporal or spatial datatypes as range of an attribute). These strategies will be discussed in detail in section 4.3.3.

Then user may commence by restructuring some part of the domain ontology to include temporal and/or spatial aspects through defining and performing Assembly Tasks. An Assembly Task can be user initiated (*i.e.*, if the user chose which concepts may be assembled and which Assembly Task is to be used) or accepted from the proposals list. Each Assembly Task (either user initiated or accepted from the proposals list) aims to create a new temporal/spatial concept by assembling an atemporal/aspatial domain concept with a temporal/spatial one.

The necessary changes to perform Assembly Tasks are codified in Assembly Rules. Before performing such restructuring Assembly Tasks a verification method (**Verify Assembly Task Consistency** ensures that the instructions specified in the Assembly Rule does not lead to an inconsistent state.

The execution of the Assembly Task (**Execute Assembly Task**) will perform changes over the target ontology and potentially generate Assembly Proposals (*e.g.*, through structural and semantic analysis or by explicit specification in the Assembly Rule). A verification method (**Ensure Assembly Proposal list Consistency**) ensures that the new Assembly Proposals are valid and search the Assembly Proposals list for proposals that became inconsistent or obsolete as consequence of the last Assembly Task execution. Ad-

ditionally, the executed Assembly Task is added to a **Historic of Assembly Tasks** so FONTE know which tasks as already been executed.

The user may subsequently decide either to perform another iteration or to go to Conclude Process and accept the current Target Ontology as the final version.

4.3 FONTE Ontology

Figure 4.3 depicts an excerpt of the FONTE ontology containing the main concepts and the most relevant relations between them.

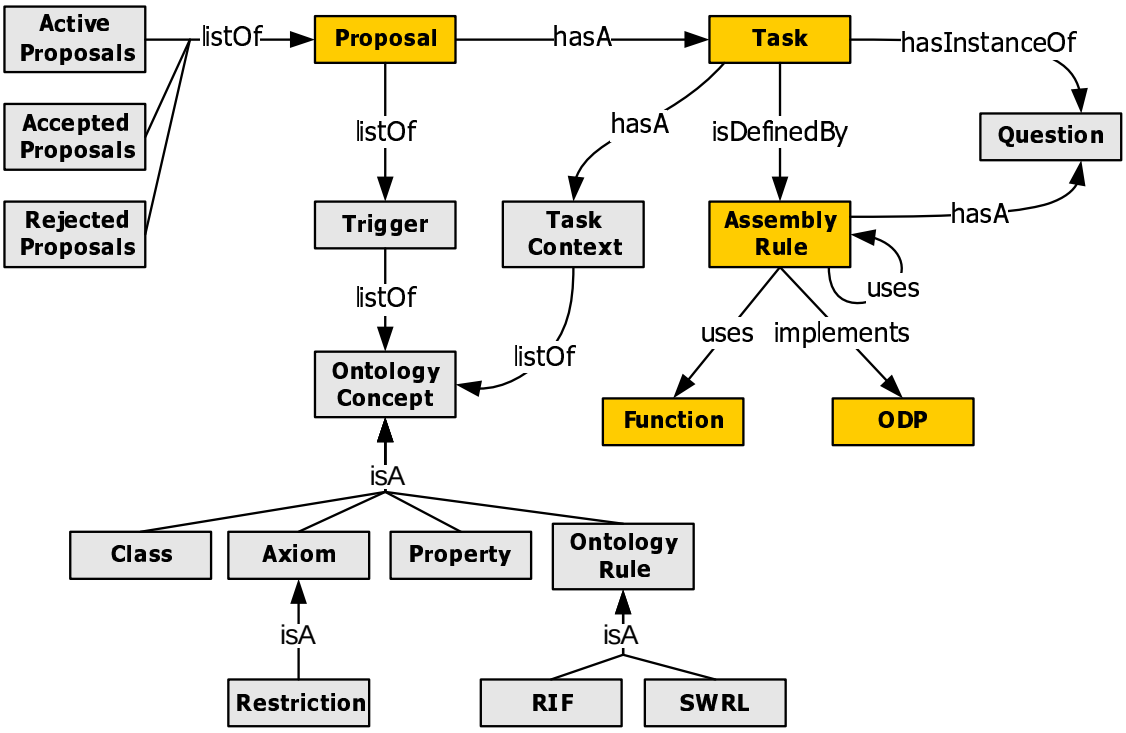


Figure 4.3: FONTE Ontology

The concept of **Assembly Task** is one of the most important concept of FONTE. A Task is the abstract concept of something that must be done in order to produce some changes in the target ontology. For instance, *defineAsEvent*/1 is an Assembly Task that allows specifying an ontology class as an event.

```
AssemblyTask defineAsEvent/1
metadata
{
    @description: "This task allows defining some class as an Event.
    An Event is something that happens at some time and in some place,
    meaning that is a spatiotemporal concept. For instance, a football
```

```
    match or a music show can be seen as an event";
}
-> assembleClass(#1,cosmo:Event)
```

At this level is not known which entities are involved or what operations should be done to perform this task. This leads to the **Assembly Task Instance** concept.

An **Assembly Task Instance** is a task for which it is already known in what context it will be executed (its input parameters), and so it is also known which Assembly Rule implementation will be invoked. For instance, if the *defineAsEvent*(foot:Match) is an Assembly Task Instance of the *defineAsEvent*/1 that will trigger the execution of the *assembleClass*(foot:Match,cosmo:Event) Assembly Rule implementation.

An **Assembly Rule** defines the set of operations that must be executed in order to perform a task. Each Assembly Rule could have several implementations, depending of the entities involved. For instance, the *assembleClass*(C1,C2) Assembly Rule can have an implementation invoked when some domain class is being defined as something with temporal extent (*assembleClass*(C1,cosmo:TemporalThing)) and a different one when being defined as an event (*assembleClass*(C1,cosmo:Event)). The Assembly Rules allows to encode ODPs.

Ontology Design Patterns are modeling solutions for recurring design problems. For instance, a common way to define a concept as an event consists in adding three subclass restrictions defining the beginning and end time points and also its spatial location. In FONTE, that ODP can be implemented using an Assembly Rule (see Algorithm 1).

rule *assembleClass*(C1,C2)
if *(C2 == cosmo:Event)* **then**
  do : addRestriction(C1,cosmo:hasStartingTimePoint,"SomeValuesFrom",
  cosmo:TimePoint,"SuperClass");
  do : addRestriction(C1,cosmo:hasEndingTimePoint,"SomeValuesFrom",
  cosmo:TimePoint,"SuperClass");
  do : addRestriction(C1,cosmo:isLocatedAtOrOn,"SomeValuesFrom",
  cosmo:GeographicalThing,"SuperClass");
end

Algorithm 1: Example of ODP implementation using Assembly Rules

The instructions defined in the Assembly Rules that allows modifying the target ontology or that allows to request the user interaction are called Assembly Functions.

An **Assembly Function** is an atomic action that should be performed in order to produce

changes in the target ontology. Functions are located at the lowest level of interaction between the user and the ontology. Each Assembly Rule interacts with the ontologies through functions. In the Algorithm 1, the *addRestriction* instruction corresponds to an Assembly Function that creates a specified ontology restriction over a particular given class. The use of functions is motivated mainly because:

- they provide an abstraction layer of communication with the API used for ontology manipulation (*e.g.*, OWL-API, Jena API) allowing the ontology engineering to model his own rules without being familiar with the underlying API;
- each function implements preconditions, ensuring that all requirements for its execution are satisfied and that after that the ontology will be in an consistent state;
- each function implements its own undo method, enabling the possibility to recover from an Assembly Rule execution;
- since the semantics associated to each Function is known it is possible to predict the impact an Assembly Rule execution.

An **Assembly Proposal** is a suggestion for the execution of a Task Instance. Each Proposal is composed by a Task Instance, a Trigger that consists in the entity that was responsible for the suggestion, a numeric value (Weight) representing the importance of that suggestion and a Natural Language Question (NLQ) that resumes in few words the objective of that suggestion. A more detailed description of proposals will be provided later in this section. A Proposal List consists in a list of proposals. There are three kinds of lists:

active proposals – is the list of proposals that could be accepted by the user;

accepted proposals – consists in a list of proposals that has been accepted by the user;

rejected proposals – that contains all the proposals that have been rejected by the user or the system itself.

The **Historic of Assembly Task Instances** is an ordered record of all executed Assembly Task Instances. This record is useful for FONTE method because: it indicates which tasks have already been executed; it allows the undo operation and; it provide statistics about the assembly process.

In the next sections it will be presented a detailed description about Assembly Task, Assembly Rule and Assembly Proposal. It will also be described the methods that are used for automatic proposal generation.

4.3.1 Assembly Task

An Assembly Task is an abstract concept of something that must be done to produce some changes in the target ontology. At this level it is not specified which changes are needed to be performed or which are the entities involved in the process. A task is identified by its name and the number of arguments, according to the following notation: *taskName*/NArgs. For instance, consider the Assembly Task *defineAsEvent*/1 which allows to define an ontology class as an event that occurs during a time period and at a specific place.

When the ontology entities involved in a Assembly Task are known it is considered an Assembly Task Instance. For instance, both *defineAsEvent*(foot:Match) and *defineAsEvent*(foot:TrainingSession) are instances of the *defineAsEvent*/1 Assembly Task.

The FONTE method recognizes three standard Assembly Tasks and assumes a particular semantics for them:

assembleClass/2 – it allows to relate one domain ontology class with one transversal temporal/spatial ontology class;

assembleRestriction/6 – it allows to assemble a restriction applied over a domain ontology class;

assembleProperty/1 – it allows to assemble a property of the domain ontology;

This is particularly useful for the automatic generation of proposals using FONTE internal proposal generation through structural and semantic analysis strategies and to perform operations that ensure the ontology consistency, such applying the inverse Assembly Task over restrictions on inverse properties (as discussed in section 3.3.2). However, it is possible to disregard this interpretation by specifying that in the assembly process setup.

The realization of an Assembly Task Instance triggers the execution of an Assembly Rule implementation, which consists an ODP implementation stating which modifications must be done in order to fulfill the task realization. The set of available Assembly Rules must

be specified in an Assembly Rule File (see section 4.3.2). Realizing Assembly Tasks allows the ontology engineering to systematically and consistently performs large amounts of modifications over the domain ontology.

4.3.2 Assembly Rule File

A brief notion of Assembly Rule has already been introduced in the previous section. In this section a more detailed analysis about the structure of Assembly Rule Files and Assembly Rules is presented. An Assembly Rule File is a set of Assembly Rules and it can be viewed as the instructions about how a particular (transversal) ontology can be assembled with any domain ontology. This means that the Assembly Rule File is absolutely independent of the domain ontology, it only concerns about a particular transversal ontology. Each Assembly Rule File consists of seven parts:

1. metainformation, as the author, the associated ontology, the date or the description;
2. the import declarations, which can be used to automatically import some ontology that will be used in the assembly process of the domain ontology (*e.g.*, the cosmo ontology);
3. the set of Assembly Tasks;
4. the set of Assembly Rules implementations;
5. the set of natural language questions, each one associated to one Assembly Rule;
6. the startup analysis rules, that are a used to produce an initial set of Proposals, which typically consists in performing semantic analysis of domain concepts and external knowledge sources;
7. the set of constraints that relate rules, which allows the definition of which tasks could or could not be performed considering the previous executed tasks.

An Assembly Rule defines the set of operations that must be executed in order to perform a task. Each Assembly Rule could have several implementations, depending of the entities involved. There must be a NLQ associated to each Assembly Rule and not to each of its implementations.

An Assembly Rule is consisted of five parts, namely:

1. Assembly Rule name;
2. input parameters (variables);
3. some optional meta-information about the rule, as the creation date, version, author or a description;
4. precondition for rule execution (an if-clause);
5. the set of rule instructions, each one associated to one special operator that defines the semantic of each action.

Each rule instruction is a pair (special operator; action). There are three special operators:

DO – try to perform the associated action;

PROPOSE – add the associated action to the active proposals list;

ASK – initiate an interaction with the user and wait for his response to continue the rule execution.

There are also three possible kinds of actions:

Function Action – as defined before, it is an atomic action that should be performed in order to produce changes in the target ontology;

Assembly Rule Action – it is the invocation of another Assembly Rule;

User Interaction Action – it displays a graphic user interface, for user interaction, in execution time.

Not all combinations between Special Operator and Action are allowed. The ASK operator can only be associated to an User Interaction action, the PROPOSE operator can only be associated to an Assembly Rule Action and the DO operator can be associated to a Function Action or to an Assembly Rule Action. These structure constraints improve the rule readability and are useful for the implementation of Assembly Rule integrity, consistency and impact prediction processes. The Assembly Rule *assembleClass*(C1,C2) (see Algorithm 2) corresponds to the rule that is invoked when an instance of the Task *assembleClass*/2 with the second parameter being the class cosmo:Role is executed.

The set of instructions of the Assembly Rule *assembleClass*(C1,C2) (see Algorithm 2) includes some DO operations, namely the function *addSubClass*/2 which creates an subclass

rule *assembleClass*(C1,C2)
if *(C2 == cosmo:Role)* **then**
  do : `addSubClass`(C1,C2);
  ask: RoleOfObject = `selectFromOntologyTree`("Is Role Of","Select the concept that represents Object played by that Role (*e.g.*, Professor is Role of Object: Person). Choose a class from the hierarchy or create a new one.");
  do :
  `addRestriction`(C1,cosmo:hasRoleFiller,"SomeValuesFrom",RoleOfObject,"SuperClass");
  do :
  `addRestriction`(RoleOfObject,cosmo:fillsTheRoleOf,"SomeValuesFrom",C1,"SuperClass");

  propose: `startTimePointOfRole`(C1),C1,45;
  propose: `endTimePointOfRole`(C1),C1,45;
end

Algorithm 2: Define class as a temporal Role

relation between the concepts C1 and C2, an ASK operation that triggers an graphical user interaction option to let the user select the object of which C1 is the role of, and two PROPOSE operations, which adds two proposals to the active proposals list suggesting the user to define the start and ending time point of the role playing.

4.3.3 Assembly Proposal

An Assembly Proposal is a suggestion for the execution of an task instance. Each Assembly Proposal is composed by the Task Instance, a Trigger, a Weight and a NLQ Instance. The Trigger specifies what triggered the Assembly Proposal generation, allowing the system to explain why that Assembly Proposal was suggested. The Weight reflects the importance/belief of the usefulness of that Assembly Proposal for the assembly process. The NLQ aims to succinctly explain the objective of the Assembly Proposal execution.

For instance, the instruction

```
NLQ for assembleClass/2 is ‘‘Define #1# as #2#?’’;
```

defines an template NLQ for the Assembly Rule *assembleClass*/2. When the Assembly Rule arguments are known the NLQ can be instantiated. For instance, the Assembly Rule *assembleClass*(foot:Match,cosmo:Event) would produce the NLQ "Define foot:Match as cosmo:Event?".

The Assembly Proposals may be generated in two ways: through FONTE method internal analysis mechanisms, which allows the creation of proposals for execution of standard Assembly Tasks (as described in section 4.3.1); or through the explicit declaration or

formulation of necessary conditions in the Assembly Rule File (as described in section 4.3.2, relatively to the PROPOSE special operator and Startup Analysis Rules, respectively).

In section 4.3.3.1 are presented several strategies for Proposals generation, according to those two approaches described earlier. Then, in section 4.3.3.2 is presented an method to ensure the Proposals List consistency.

4.3.3.1 Assembly Proposals Generation Strategies

In this section several strategies for generation of Proposals are presented.

Semantic Analysis of the Domain Ontology – The FONTE method implements mechanisms for similarity measure through semantic analysis of the domain ontology entities. After the execution of one instance of the *assembleClass*/2 generic Assembly Task the system tries to find similar domain classes and propose them to also incur in changes. For instance, if after the execution of the *assembleClass*(foot:City,cosmo:GeographicalRegion) Assembly Task Instance the system would find similar concepts such foot:Country and foot:Continent and propose them to be assembled with cosmo:GeographicalRegion. The implemented semantic similarity analysis process is based in a study performed by Xu et al. [Xu et al., 2008], which consider five factors to calculate the similarity factor, namely:

1. local density;
2. node depth;
3. link type;
4. link strenght;
5. node attribute.

This process could be enhanced by considering label annotations (*e.g.*, rdf:label) and using wordnet to compare the classes labels and infer implicit relations between them.

Internal FONTE analysis strategies – As said before, the FONTE method assumes the existence of three standard Assembly Tasks (*assembleClass*/2, *assembleRestriction*/6 and *assembleProperty*/1) with particular semantics. The execution of a instance of one of these tasks could create new proposals:

- by executing an instance of *assembleClass*/2 task the system may propose the assemble of all restrictions of the participant domain class;

- by executing an instance of *assembleRestriction*/6 task the system may propose the assemble of the restriction property and the assemble of the object participant class. For instance, if the restriction foot:FootballPlayer foot:hasTeam *only* foot:Team is defined as temporal, then both foot:FootballPlayer and foot:Team are proposed to become temporal;
- by executing an instance of *assembleProperty*/1 Assembly Task the system may propose the assemble of its domain and range classes.

Explicit declaration in the Assembly Rule – The user can state that a specific Task Instance will be proposed by declare it in the set of instructions of an Assembly Rule, through the PROPOSE special operator. By doing that the user is explicitly saying that that Proposal should be created if that Assembly Rule implementation is executed. This is useful to provide proposals for specific domain tasks. For instance, the Assembly Rule that is triggered when some class is assembled to become temporal or spatial can contain instructions to generate proposals to define its temporal start and ending time points or the spatial location, respectively.

Use of External Knowledge Source – The FONTE method provides an useful mechanism to reuse knowledge existing in external sources, usually upper ontologies. The main idea is to use that external knowledge to infer new proposals by specifying the necessary conditions for its creation in the Assembly Rule File.

For instance, consider the SUMO[1] (Suggested Upper Merged Ontology) upper ontology. This ontology describes, among others, temporal and spatial concepts (*e.g.*, sumo:SocialRole and sumo:Region). If the user state that the domain ontology class foot:President is similar to the class sumo:President, that the class sumo:President is a sub class of sumo:SocialRole and that sumo:SocialRole is equivalent to cosmo:Role, its reasonable to suggest that foot:President might be a sub class of cosmo:Role. This result in the creation of the proposal *assembleClass*(foot:President,cosmo:Role);

Analysis of the involved Data Types – OWL 2 provides support for use of datatypes available in XML Schema Definition Language (XSD) 1.1. Although XSD 1.1 is still not a W3C Recommendation and the use of its datatypes in OWL is optional, it could be used to explore some implicit semantics. For instance, XSD 1.1 defines some temporal datatypes (*e.g.*, dateTimeStamp, duration, gDay, gMonth,

[1] `www.ontologyportal.org`

gYear), which could mean that the properties that use these datatypes as its range could be considered temporal properties and should be proposed to be assembled. Consequently, all restrictions that use these properties, and classes that implements those restrictions, could also be proposed to be assembled;

Lexical Analysis of the involved Property – There are several words or expressions that reflect temporal and spatial notions. For example, the words "location" or "duration" are usually used to describe entities with spatial location or temporal extent. These kind of words used to identify properties (or associated to them as annotations), could be analyzed to propose new Assembly Tasks.

4.3.3.2 Proposals List Consistency

During the Assembly Process many proposals may be generated. However, the modifications performed over the ontology may turn these proposals obsolete or even incorrect.

Obsolete Proposals – are those which acceptance will not add anything new to the ontology, since all the modifications that its acceptance would provoke were already undertaken.

Incorrect Proposals – are those which acceptance will produce an inconsistency to the ontology.

It was created a mechanism to establish relations between Assembly Proposals. The two possible relations between Assembly Proposals are contains and disjoint.

contains – A proposal Prop_1 contains a proposal Prop_2 when all modifications that result from accepting the Prop_2 also result from accepting Prop_1. For instance, consider the Assembly Task *assembleClass*/2. For simplicity the execution of this task simply creates an subClass relation between the two argument classes. Now consider three classes A, B and C from which B is a sub class of A, and C is not related to any of them. The system as created two proposals Prop_1 and Prop_2 for class assembly, namely *assembleClass*(A,C) and *assembleClass*(B,C). Since by executing the *assembleClass*(A,C) Assembly Task the class A becomes sub class of C, because of the transitivity of the subClass property the class B also becomes subclass of C. That is, the acceptance of the *assembleClass*(B,C) will not create any new knowledge.

disjoint – A proposal Prop_1 is disjoint from a proposal Prop_2 when the execution

of one of them invalidates the execution of the other. For instance, consider that the system as generated two proposals $Prop_1$ and $Prop_1$ for class assembly, namely *assembleClass*(A,B) and *assembleClass*(A,C). Now consider that the classes B and C are disjoint from each other. The acceptance of both proposals would result in the creation of a subClass relation between the classes A and B and the classes A and C. Since the classes B and C were defined as disjoint from each other, this would result in an ontology inconsistency.

Since an Assembly Proposal is a system suggestion for an Assembly Task execution and that the necessary modifications to perform an Assembly Task are defined by an Assembly Rule, then the possible relations between proposals depend of which Assemble Rule is triggered by the Assembly Proposal execution.

FONTE is able to automatically infer the relations between Assembly Proposals by analyzing and comparing the content of the Assembly Rule triggered by each proposal. FONTE only considers the instructions that result in the execution of functions that change structure of the target ontology (*i.e.*, those defined using the DO special operator, as presented in section 4.3.2). Since the impact that each function execution is well known by FONTE it is possible to compare and establish the relation between each Assembly Task execution and consequently between each Assembly Proposal.

However, there are limitations in this approach. The execution of some Assembly Tasks require user action to drive the process and so it is impossible to predict which modifications will be performed. Because of that this method only considers rules whose actions are possible to predict.

Nevertheless, FONTE also allows the user to specify the relations between proposals by specifying the set of constraints that relate rules. This is particularly useful if the user knows that some Assembly Task must not be executed after another because of some domain constraints and not since its execution would result in an inconsistency.

4.4 Protégé Plug-in

Protégé is one of the most widely used open source ontology editor and knowledge-base framework, which provides a powerful graphic interface. In order to support the iterative and interactive process used in FONTE a Protégé plug-in (version 4.1) was developed. This

plug-in provides a set of functionalities, such as:

- to setup the FONTE method by choosing the Assembly Rule File and choosing if the system may consider the standard Assembly Tasks;
- to apply ODPs over ontology concepts such as classes, properties and restrictions by choosing one from the ODPs list;
- to manage the lists of Assembly Proposals:
  - to accept or reject a proposal execution from the active proposals list;
  - to filter the proposal list according to severel criteria, namely the Assembly Task and the involved ontology concepts;
  - to visualize the relations of specific proposal with others;
  - to visualize the relations of all proposals;
  - to visualize which proposals as been rejected and accepted;
- to know which modifications are being performed by analyzing the FONTE activity log;
- to undo an user action execution (regardless if it consists on applying an ODP or accepting/rejecting a proposal);
- and to visualise statistics of the assembly process.

As presented in Figure 4.4, the plug-in presents two panels: one for the manipulation of the participant ontologies (in the left-hand side) and the other for the lists of proposals (in the right-hand side).

The panel further to the left contains the domain ontology (football ontology, which is timeless and spaceless); from this panel it is possible to access the classes and object/data properties hierarchies. The other panel contains the temporal/spatial ontologies to be used as construction blocks for the production of the target ontology.

Below these two hierarchy panels there is a description of the selected domain class. This description consists of the equivalent and super classes, and other information such the disjoint classes and the members (the instances of the class). Through this panel it is possible to select an restriction to assembly.

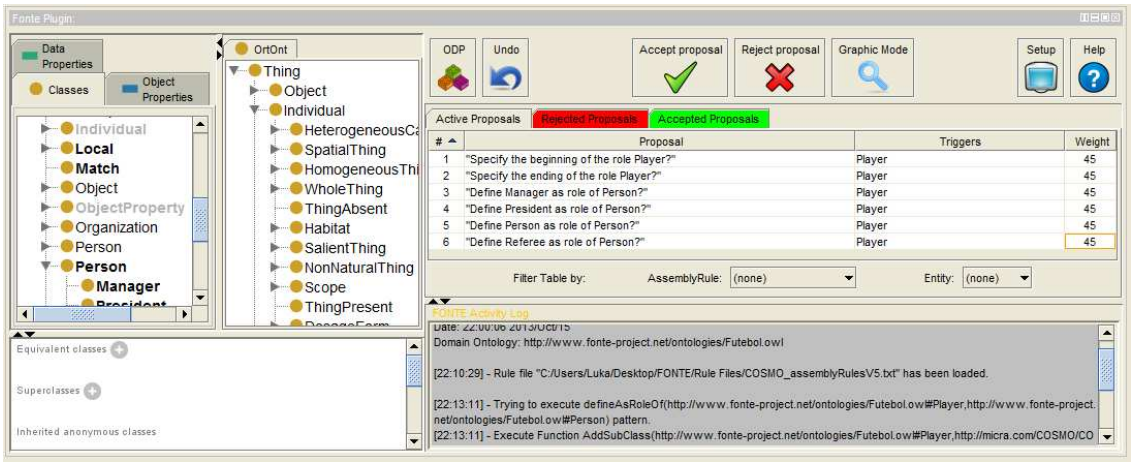


Figure 4.4: FONTE Protégé plug-in

The lists of proposals consists of three lists, containing the active, rejected and accepted proposals. It is possible to sort the lists by number, name, trigger or weight. It it also possible to filter the list by Assembly Task and/or ontology concept involved.

Below the lists of proposals there are the FONTE activity log panel, in which all operations that are being performed are described, including the description of each Assembly Task execution and explanations for possible Assembly Task execution errors.

Above the lists of proposals there are several buttons. The Task button allows the user to select and apply an AssemblyTask (see Figure 4.5). The Undo button allows the user to undo some operation, regardless it consists on applying an Assembly Task or accepting/rejecting a proposal.

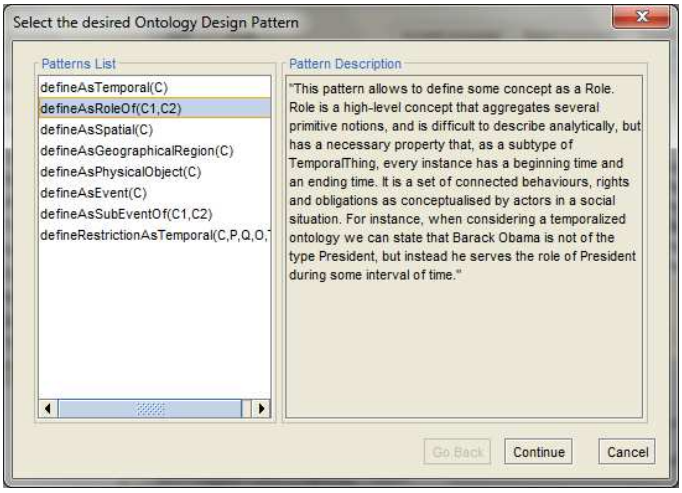


Figure 4.5: Choosing an Assembly Task

The Accept Proposal and Reject Proposal buttons allows to accept and/or reject a proposal or a list of proposals selected from the Active Proposals List. The Graphic Mode button displays an graph that consists of the relations between the proposals of the Active Proposals List.

The Setup button (see Figure 4.6) allows to user to choose the Assembly Rule File that will be user to guide the Assembly Process and to choose if FONTE must consider the three standard Assembly Tasks *assembleClass*/2, *assembleRestriction*/6 and *assembleProperty*/1 when performing internal mechanisms for generating proposals.

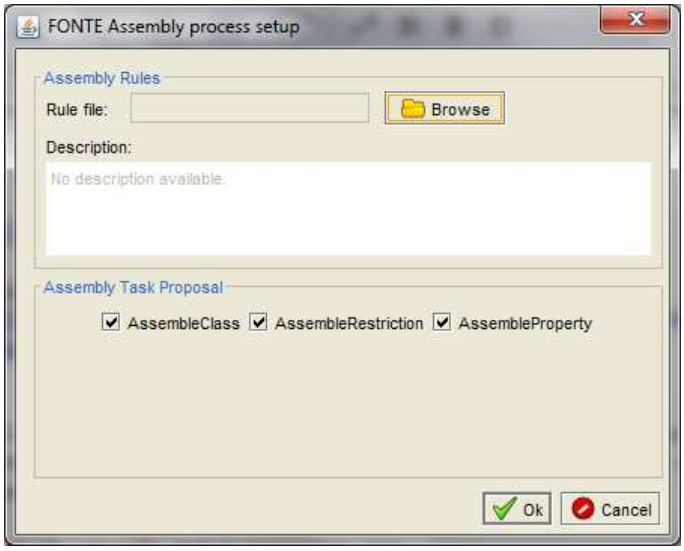


Figure 4.6: Setting up FONTE

All the Assembly Tasks that are successfully performed (either triggered manually by user-driven action or by accepting on of the active proposals suggested by the system) are added to a list containing the Assemble Tasks history.

Associated to each Assembly Proposal there is a question in natural language, a trigger list and the task weight. The question in natural language is composed by a phrase that summarises the proposal objective, instantiated with the elements contained in the instance task. The trigger list is composed by the elements that triggered the proposal. The weight provides an indication of the importance of each proposal; the higher the weight, the higher the possibility of the proposal to be accepted during the assembly process.

As the assembly process progresses, more proposals are generated. If different concepts happen to propose the same task instance, all the elements that have triggered that

proposal are included in the trigger list and the proposal weight is increased to reflect its relevance. In order to avoid overloading the knowledge engineer with unuseful proposals, the system defines that a proposal that has been rejected before can not be proposed again. The proposals are therefore filtered by an auto-rejection system. However, it should be noted that the knowledge engineer has the ability to recover a rejected proposal.

4.5 Temporal and Spatial Engineering with FONTE

In order to demonstrate the assembly process supported by FONTE method it will be presented a running example of how to include temporal and / or spatial knowledge in domain ontologies. It will be used both COSMO and Football ontologies, previously presented (see sections 3.1 and 3.2). The engineering process will follow the endurantist approach as presented in section 2.1.1. To support the FONTE method, several ODP have been codified using Assembly Rules. These ODPs include:

1. Defining a class as something with temporal extent;
2. Defining a class as an event;
3. Defining a class as a temporal role played by something;
4. Defining a class as something spatial;
5. Defining a class as a geographical region;
6. Defining a class as a physical object;
7. Defining a class restriction as temporal;

4.5.1 Defining a class as a temporal role played by something.

The assembly of classes is usually the first step of the assembly process. This process concerns the operations that need to be applied, in order to produce the desired changes in the target domain class. Usually, this regards the addiction or removal of relations. For instance, the class foot:Player will be defined as a temporal role played by something; in this case, an instance of the class foot:Person. This is done by applying the defineAsRoleOf ODP over both classes: *defineAsRoleOf*(foot:Player,foot:Person). This will trigger the *defineAsRoleOf*(foot:Player,foot:Person) Assembly Rule (see Algorithm 3).

This rule starts by creating a subclass relation between the class foot:Player and the class cosmo:Role. Then it adds a restriction stating that every instance of foot:Player may have a

```
rule defineAsRoleOf(C1,C2)
if () then
    do : addSubClass(C1,cosmo:Role);
    do : addRestriction(C1,cosmo:hasRoleFiller,"SomeValuesFrom",C2,"SuperClass");
    propose: startTimePointOfRole(C1),C1,45;
    propose: endTimePointOfRole(C1),C1,45;
    for SimClass : getSimilarClasses(C1,0.6) do
        propose: defineHasRoleOf(SimClass,C2),C1,45;
    end
    do : removeSubClass(C1,C2);
end
```

Algorithm 3: Define some class as a temporal Role of another

#	Proposal	Trigger	Weight
1	"Define the starting time point of the foot:Player role playing?"	foot:Player	45
2	"Define the ending time point of the foot:Player role playing?"	foot:Player	45
3	"Define foot:Person as a role played by foot:Person?"	foot:Player	45
4	"Define foot:Manager as a role played by foot:Person?"	foot:Player	45
5	"Define foot:Referree as a role played by foot:Person?"	foot:Player	45
6	"Define foot:Persident as a role played by foot:Person?"	foot:Player	45

Table 4.1: Active Proposals List at step #1

relation with an instance of the class foot:Person through the property cosmo:hasRoleFiller. Then it creates two proposals, namely the suggestion for the user to specify the start and end time points of the role playing. After that, the system searches for all classes that are semantically similar to foot:Player, by using a semantic similarity algorithm [Xu et al., 2008]. Finally, if exists some subclass relation between the classes foot:Player and foot:Role it is removed. As result of the rule execution, six proposals were created, as presented in Table 4.1.

4.5.2 Rejecting a proposal

By analyzing the list of proposals it plausible to say that the proposal number 3 is not valid, since being a person is not a role played by a person. For that reason the proposal will be rejected, which consists in add that proposal to the rejected proposals list. This means that that specific proposal will never be proposed again during the assembly process.

#	**Proposal**	**Trigger**	**Weight**
1	"Define the starting time point of the foot:Player role playing?"	foot:Player	45
2	"Define the ending time point of the foot:Player role playing?"	foot:Player	45
5	"Define foot:Referree as a role played by foot:Person?"	foot:Player, foot:Manager	90
6	"Define foot:Persident as a role played by foot:Person?"	foot:Player, foot:Manager	90
7	"Define the starting time point of the foot:Manager role playing?"	foot:Manager	45
8	"Define the ending time point of the foot:Manager role playing?"	foot:Manager	45

Table 4.2: Active Proposals List at step #3

4.5.3 Accepting a proposal

Carrying on the assembly of classes, the proposal 4 will be accepted, since being a Manager is also a role played by a person. The acceptance of the proposal 4 will trigger the execution of the *defineAsRoleOf*(foot:Manager,foot:Person) Assembly Task, similarly to what happened in the previous task. The proposal to define the class foot:Person as a role played by a foot:Person will not be generated this time despite foot:Person class is semantically similar to person:Manager class, since that proposal was previously rejected. The class foot:Player is also semantically similar to foot:Manager, but that class has already been defined as role of foot:Person, so the system will not create that proposal again.

This time, only 4 proposals will be generated: two of them concerning the start and ending time points of the foot:Manager role playing and two of them concerning the semantic similar classes of foot:Manager, namely foot:Referree and foot:President. The last two already exist in the list of proposals, so their trigger and weight will be updated. At this point the list of active proposals is as presented in Table 4.2.

4.5.4 Accepting multiple proposals

The assembly of classes continues by accepting the proposals number 5 and 6. The changes in the ontology and the generated proposals will proceed analogously to what happened in step 3. At this point, the classes foot:Player, foot:Manager, foot:Referee and foot:President has been defined as roles. Several changes have been performed over the domain ontology by simply executing an Assembly Task.

There are currently 8 proposals in the proposals list, concerning the specialization of the start and ending time points of the roles playing. For instance, the foot:Player role playing can start with the event of signing the contract. The user may specify it by accepting the proposal 1. Although the concept of contract is not part of the presented Football ontology, that concept can be created during the acceptance of the proposal, since one of the actions of the rule associated to the proposals 1 is the choice of the class that must initiate the role playing, or the creation of a new one. However, the referred proposals will be rejected since their acceptance is not crucial and increases the ontology complexity by creating new concepts. This is merely a modeling option.

4.5.5 Defining a class as an event

The assembly process continues with the choice of another class to assemble. This time the class foot:Match will be characterized as an event. This is done by applying the *defineAsEvent*/1 Assembly Task over the foot:Match class. This action triggers the execution of the Assembly Rule *assembleClass*(foot:Match,cosmo:Event) according to the Algorithm 4.

```
rule assembleClass(C1,C2)
if (C2==cosmo:Event) then
    do : addSubClass(C1,C2);
    propose: defineEventLocation(C1),C1,45;
    for SimClass : getSimilarClasses(C1,0.6) do
        propose: assembleClass(SimClass,C2),C1,45;
    end
end
```

Algorithm 4: Define some class as an Event

This Assembly Rules starts by creating an subclass relation between the classes foot:Match and cosmo:Event. Consequently, the class foot:Match inherits the temporal and spatial characteristics of the cosmo:Event class, namely the restrictions about the start and ending time points and the spatial location of the occurrence. Additionally it creates a proposal that allow the user to specify a more precise location of the foot:Match event. It also searches for potential similar classes of foot:Match in order to propose them to be characterized as events; in this case no similar classes were found. At this point the list of active proposals is as presented in Table 4.3.

The proposal 13 will be accepted in order to define the location of the foot:Match event. By accepting the proposal 13 the Assembly Rule *defineEventLocation*(foot:Match) is invoked (see Algorithm 5).

#	Proposal	Trigger	Weight
13	"Define the location where the event foot:Match occurs?"	foot:Match	45

Table 4.3: Active Proposals List at step #5

rule *defineEventLocation*(C1)
if *()* **then**
  ask: EventLocation = `selectFromOntologyTree`("Event Location","Select the concept that is the location where this event occurs. Choose a class from the hierarchy or create a new one.");
  do : `assembleClass`(EventLocation,cosmo:SpatialThing);
  do : `addRestriction`(C1,cosmo:occurredAt,"SomeValuesFrom",EventLocation, "SuperClass");
  do : `addRestriction`(EventLocation,cosmo:wasTheLocationOf,"SomeValues From",C1,"SuperClass");
end

Algorithm 5: Define location for some Event

The Assembly Rule execution starts by requesting the user to select (or create) a class that corresponds to the location where the event foot:Match occurs. In this case, the class foot:Stadium was selected. The next rule action defines that the selected location of the event must be a Spatial Thing, which is ensured by invoking the rule *assembleClass*(foot:Stadium,cosmo:SpatialThing). Then it creates a restriction stating that the event foot:Match occurs in a foot:Stadium and the inverse relation stating that the foot:Stadium is the location of the event foot:Match.

4.5.6 Defining a class as a Spatial Thing

In the step 5 of the assembly process, the class foot:Stadium has been defined as a Spatial Thing by executing the Assembly Rule *assembleClass*(foot:Stadium,cosmo:SpatialThing), as presented in the Algorithm 6.

rule *assembleClass*(C1,C2)
if *(C2) == cosmo:SpatialThing* **then**
  do : `addSubClass`(C1,C2);
  propose: `defineLocation`(C1),C1,60;
  propose: `specializeSpatialThing`(C1),C1,45;
  for *SimClass : getSimilarClasses(C1,0.6)* **do**
    propose: `assembleClass`(SimClass,C2),C1,45;
  end
end

Algorithm 6: Define some class as a Spatial Thing

This rule starts by classify the class foot:Stadium as cosmo:SpatialThing by creating a subclass relation between them. Then two proposal to define the location of the Stadium

#	Proposal	Trigger	Weight
14	"Define the location of foot:Stadium?"	foot:Stadium	45
15	"Define a more specific type of Spatial Thing for foot:Stadium?"	foot:Stadium	45

Table 4.4: Active Proposals List at step #6

and to set a more precise spatial classification of foot:Stadium are added to the proposals list. Also, FONTE searches for foot:Stadium similar classes in order to propose them to be classified as spatial things; in this case no similar classes were found. At this point the list of active proposals is as presented in Table 4.4.

As consequence of the classification of foot:Stadium as a Spatial Thing, the user can specialize this classification by choosing a more specific one, namely by classifying it as a Physical Object or a Geographical Region. This can be done by accepting the proposal 15, which triggers the execution of the Assembly Rule *specializeSpatialThing*(foot:Stadium).

rule *specializeSpatialThing*(C1)
if *()* **then**
  ask: Answer = `MultipleChoice`("Specialize Spatial Thing","Is this spatial thing a Physical Object or a Geographical Region?",["Physical Object","Geographical Region"]);
  if *Answer == "Physical Object"* **then**
    do : `assembleClass`(C1,cosmo:PhysicalObject);
  end
  if *Answer == "Geographical Region"* **then**
    do : `assembleClass`(C1,cosmo:GeographicalRegion);
  end
end

Algorithm 7: Specialize Spatial Thing as Physical Object or Geographical Region

During the execution of Assembly Rule *specializeSpatialThing*(foot:Stadium) the user is questioned if the class consists of a physical object or a geographical region. In this case the first option will be taken and consequently the rule *assembleClass*(foot:Stadium,cosmo:PhysicalObject) will be invoked, which leads to the step #7.

4.5.7 Defining a class as a Physical Object

The execution of the Assembly Rule *assembleClass*(foot:Stadium,cosmo:PhysicalObject) (see Algorithm 8) is started by classifying the class foot:Stadium as cosmo:PhysicalObject. Since, by definition, every physical object in cosmo is a temporal thing, the rule *assembleClass*(foot:Stadium,cosmo:TemporalThing) is invoked. Then two proposals to define the location of the class foot:Stadium are created. The first allows defining a more generic

location for the class. The second enables the user to state that every foot:Stadium as a GPS location. Also, FONTE searches for foot:Stadium similar classes in order to propose them to be classified as physical objects; in this case no similar classes were found.

```
rule assembleClass(C1,C2)
if (C2 == cosmo:PhysicalObject) then
    do : addSubClass(C1,C2);
    do : assembleClass(C1,cosmo:TemporalThing);
    propose: defineLocation(C1),C1,45;
    propose: setGPSLocation(C1),C1,30;
    for SimClass : getSimilarClasses(C1,0.6) do
        propose: assembleClass(SimClass,C2),C1,45;
    end
end
```

Algorithm 8: Define some class as a Physical Object

Since the proposal to define the location of foot:Stadium already exists in the proposals list and triggered by the foot:Stadium concept, the proposal 14 remains unchanged (see Table 4.5).

The acceptance of the proposal 16 leads to the execution of the Assembly Rule setGPSLocation(foot:Stadium). This assembly rule consists in create 3 restrictions over the foot:Stadium, one for each GPS component, namely altitude, latitude and longitude, as presented in Algorithm 9.

```
rule setGPSLocation(C1)
if () then
    do : addRestriction(C1,cosmo:hasAltitude,"SomeValuesFrom",
    cosmo:AltitudeValue,"SuperClass");
    do : addRestriction(C1,cosmo:hasLatitude,"SomeValuesFrom",cosmo:Latitude,
    "SuperClass");
    do : addRestriction(C1,cosmo:hasLongitude,"SomeValuesFrom",
    cosmo:Longitude,"SuperClass");
end
```

Algorithm 9: Set the GPS Location of a Physical Object

The acceptance of the proposal 14 results in the execution of the Assembly Rule defineLocation(foot:Stadium) as presented in Algorithm 10.

The Assembly Rule execution starts by questioning the user about the nature of the

#	**Proposal**	**Trigger**	**Weight**
14	"Define the location of foot:Stadium?"	foot:Stadium	45
16	"Set a GPS location for foot:Stadium?"	foot:Stadium	45

Table 4.5: Active Proposals List at step #7

rule *defineLocation*(C1)
if *()* **then**
  ask: Answer = `multipleChoice`("Define Location","Is this located on some Physical Object or at some Geographical Region?",["Physical Object","Geographical Region"]);
  if *(Answer == "Physical Object")* **then**
    ask: ObjectContainer = `selectFromOntologyTree`("Location","Select de object that contains this one (*e.g.*, The pencil is located at the pencil box). Choose a class from the hierarchy or create a new one.");
    do : `addRestriction`(C1,cosmo:isLocatedAt,"SomeValuesFrom", ObjectContainer,"SuperClass");
    do : `addRestriction`(ObjectContainer,cosmo:isTheLocationOf,"Some ValuesFrom",C1,"SuperClass");
    do : `assembleClass`(ObjectContainer,cosmo:PhysicalObject);
  end
  if *(Answer == "Geographical Region")* **then**
    ask: GeographicalLocation = `selectFromOntologyTree`("Location","Which is the location of the concept? Choose a class from the hierarchy or create a new one.");
    do : `addRestriction`(C1,cosmo:isLocatedAt,"SomeValues-From",GeographicalLocation,"SuperClass");
    do : `addRestriction`(GeographicalLocation,cosmo:isTheLocationOf,"Some-ValuesFrom",C1,"SuperClass");
    do : `assembleClass`(GeographicalLocation,cosmo:GeographicalRegion);
  end
end

Algorithm 10: Define the location of some Spatial Thing

location of the class foot:Stadium, namely if the it is located in some Physical Object or in some Geographical Region. In this case, the Geographical Region option will be selected. By doing that the user action is requested again, this time to select the class that corresponds to the location of foot:Stadium. The class foot:City will be selected. Two new restrictions are created, one stating that foot:Stadium is located at a foot:City and other stating that foot:City is the location of foot:Stadium. Additionally the class foot:City is classified as cosmo:GeographicalRegion through the execution of the Assembly Rule *assembleClass*(foot:City,cosmo:GeographicalRegion).

4.5.8 Defining a class as a Geographical Region

Defining foot:City as a Geographical Region is done by invoking the Assembly Rule *assembleClass*(foot:City,cosmo:GeographicalRegion) (see Algorithm 11).

The Assembly Rule execution starts by creating a subclass relation between the classes foot:City and cosmo:GeographicalRegion. Similarly to cosmo:PhysicalObject, every instance of cosmo:GeographicalRegion is, by definition, a temporal thing. Therefore the Assembly Rule *assembleClass*(foot:City,cosmo:TemporalThing) is executed. A proposal to define

```
rule assembleClass(C1,C2)
if (C2 == cosmo:GeographicalRegion) then
    do : addSubClass(C1,C2);
    do : assembleClass(C1,cosmo:TemporalThing);
    propose: defineLocation(C1),C1,45;
    for SimClass : getSimilarClasses(C1,0.6) do
        propose: assembleClass(SimClass,C2),C1,45;
    end
end
```

Algorithm 11: Define some class as a Geographical Region

#	**Proposal**	**Trigger**	**Weight**
17	"Define the location of foot:City?"	foot:Stadium	45
18	"Define foot:Country as cosmo:GeographicalRegion?"	foot:City	45
19	"Define foot:Continent as cosmo:GeographicalRegion?"	foot:City	45
20	"Define foot:Local as cosmo:GeographicalRegion?"	foot:City	45

Table 4.6: Active Proposals List at step #8.1

the location of the geographical region foot:City is created. Finally, the FONTE method searches for foot:City similar classes in order to propose them to also be classified as Geographical Regions. This results in the creation of three proposals for the characterization of the classes foot:Country, foot:Continent and foot:Local as Geographical Regions, since they are similar to the class foot:City. At this point the list of active proposals is as presented in Table 4.6.

Both proposals 18 and 19 were accepted, which results in the creation of two additional proposes for the definition of the location of foot:Country and foot:Continent. The proposal 20 was rejected by users' choice. After these actions the active list is as shown in Table 4.7

The acceptance of each available proposal will trigger the execution of the Assembly Rule *defineLocation*/1 as previously explained.

#	**Proposal**	**Trigger**	**Weight**
17	"Define the location of foot:City?"	foot:Stadium	45
21	"Define the location of foot:Country?"	foot:Country	45
22	"Define the location of foot:Continent?"	foot:Continent	45

Table 4.7: Active Proposals List at step #8.2

4.5.9 Defining class restriction as temporal

At this point it is assumed that the temporal/spatial characterization of classes is finished. The focus will now turn to the class restrictions. For instance, the class restriction stating that a Player can have a relation to a team through the property foot:hasTeam is a candidate to be temporalized, since a Player can have several teams during his career. Hence, the *defineRestrictionAsTemporal*/5 Assembly Task is applied over the referred restriction. This will trigger the execution of the Assembly Rule *assembleRestriction*(foot:Player,foot:hasTeam,"AllValuesFrom",foot:Team,"SuperClass") (see Algorithm 12).

```
rule assembleRestriction(C1,P,Q,C2,RT)
if isObjectProperty(P) then
    do : Token = createConceptIdentifier(C1,"_",P,"_",C2,
    "_TemporalRelation");
    do : createClass(Token,cosmo:TemporalRelation);
    do : addRestriction(Token,varP,Q,C2,"SuperClass");
    do : addRestriction(Token,cosmo:hasStartingTimePoint,"SomeValuesFrom",
    cosmo:TimePoint,"SuperClass");
    do : addRestriction(Token,cosmo:hasEndingTimePoint,"SomeValuesFrom",
    cosmo:TimePoint,"SuperClass");
    do : replaceRestriction(C1,P,Q,C2,P,Q,Token,RT);
    propose: assembleClass(C1,cosmo:TemporalThing),P,50;
    propose: assembleClass(C2,cosmo:TemporalThing),P,50;
end
```

Algorithm 12: Define class restriction as temporal

As presented in section 3.3.2, the temporal characterization of relations implies the creation of a new class to play the role of N-ary relation. In FONTE Assembly Rules this is done by performing the *createConceptIdentifier*/*n* function that creates a string by concatenating all arguments, followed by the *createClass*/1 function that takes the generated String and creates a class with that name. Optionally, *createClass*/2 function can be used to indicate a super class. In this case the string foot:Player_hasTeam_Team_TemporalRelation is generated and a class with that name is created has subclass of cosmo:TemporalRelation. Then two restrictions stating the start and ending time points of the new class are created, since the class cosmo:TemporalRelation does not contain them. Afterwards a restriction relating the new class to the class foot:Team is created and the original restriction is replaced to point to the new class instead of to foot:Team. Additionally two proposals are created, suggesting the classification of foot:Player and foot:Team as temporal things.

Since the class:Team has a restriction that relates it with the class:Player through the

property foot:teamOf and that foot:teamOf is inverse property of foot:hasTeam the FONTE method internal mechanism triggers the execution of the *assembleRestriction*/5 AssemblyRule over that restriction. This ensures the consistency of the inverse restrictions.

The same Assembly Task is applied to the restriction that relates the class foot:Manager to the class foot:Team over the property foot:managerOf.

Chapter 5

Results and Discussion

In this chapter will be presented and discussed the results obtained when using FONTE for engineering time and space in a timeless and spaceless ontology previously presented, the "Football Ontology". More specifically, it will be presented statistical data about the assembly process such as the number modifications performed in the ontology compared to the number of Assembly Tasks performed by the user. It will also be presented an analysis on how FONTE helped the user to carry out the operations by analyzing the amount of Assembly Tasks initiated by the user and the tasks proposed by the system.

In section 5.2 it is shown how FONTE can be used to allow the engineering of time and space using different ontologies than COSMO and even other theories than the endurantism for modeling time and space. Two Assembly Rules concerning the engineer of classes and restrictions using the 4DFluents ontology and the perdurantist theory are presented.

The possibility of using FONTE method to perform operations between ontologies other than engineering time and space in OWL domain ontologies is discussed in section 5.3.

5.1 Case Study Results

The classes foot:Player, foot:Manager, foot:Referee and foot:President were defined as roles played by instances of the class foot:Person. The class restriction between foot:Player and foot:Team through the property foot:hasTeam was defined as temporal, as well as the inverse restriction through the property foot:teamOf that exists in the opposite direction. The same modification was performed over the class restriction between foot:Manager and foot:Team through the property foot:managerOf.

The class foot:Match has been defined as an event that must occur in a foot:Stadium. In

turn, foot:Stadium was classified as an physical object that must be localized in a foot:City.

The classes foot:City, foot:Country and foot:Continent were classified as geographical regions. Also, restrictions specifying that every city is located at a country and every country is located at a continent were created.

The engineering process resulted in the creation of 4 new classes and 48 restrictions and also the removal of 6 restrictions[1]. These results are directly related to the user choices for temporal and spatial engineering. It is plausible to affirm that different users may come to different results.

During the assembly process the user had to take the initiative to choose which Assembly Task to apply only four times, namely to define:

1. foot:Player as a role of foot:Person;
2. foot:Match as an cosmo:Event;
3. restriction foot:FootballPlayer foot:hasTeam *only* foot:Team as temporal;
4. restriction foot:Manager foot:managerOf *only* foot:Team as temporal.

All other changes have arisen as a consequence of the application of those four Assembly Tasks or by system suggestion. More precisely, the system as made 32 proposals of which 11 were accepted, 8 were postponed for further analysis and 13 were rejected by users decision. Additionally, the FONTE proposal list consistency system was able to filter the proposal of 15 Assembly Tasks. These statistics are presented in Table 5.1.

Assembly Tasks	Amount
Initiated by the user	4
Proposed by system	32
Accepted proposals	11
Proposals postponed for further analysis	8
Proposals rejected by user	13
Proposals automatically rejected by FONTE	15

Table 5.1: Summary of the Assembly Process statistics

As has been shown the assembly process involved 58 changes to the domain ontology by simply invoke the execution of 15 Assembly Tasks and resulted in a consistent ontology.

[1]The classes, properties and restrictions that resulted from using the temporal and spatial ontology are not considered here since them are not relevant in analyzing the impact of using FONTE.

The effort required to manually perform these changes would be much higher. The user would have to know and understand the temporal and spatial theories and the used ontology. Then the user need to know which set of modifications he has to realize in order to define a class, a property or a restrictions as temporal/spatial and effectively apply them. Considering that the user may need to create complex structures (*e.g.*, to support the specification of temporal class restrictions) and that the modifications performed can result in an inconsistent ontology, the user need to be extremely careful. Additionally, there are the possibility that the user does not temporally/spatially characterize some concept that he wanted to because he forgets it or some concept that he have not considered initially.

5.2 Considering different ontologies and/or theories

The temporal and spatial engineering methodology presented in this document arbitrarily considers the endurantist point of view of modeling time and space and the presentation of the method was supported by a case study that considered the COSMO ontology. These assumptions were also followed in the presentation of the FONTE method and tool, for which was developed a set of Assembly Rules regarding the COSMO ontology.

However, the method is flexible enough to support different ontologies of time and space or different theories of modeling temporal and spatial knowledge (*e.g.*, perdurantist view). For instance, consider the 4D-fluentsDEN ontology about time and space presented in [Batsakis and Petrakis, 2011] which follows the perdurantist theory. By changing the set of Assembly Rules the FONTE method is able to support the temporal and spatial engineering using that ontology and approach. The prefix "4Dfluents:" will be used to refer to the namespace 4D-fluents ontology namespace `http://www.intelligence.tuc.gr/~batsakis/ontologies/4d-fluentsDEN.owl`.

For instance, the Assembly Rule *assembleClass*(C1,C2) described by the Algorithm 13 can be used to define a class as temporal by creating a new class that will play the role of the first and by defining the necessary class restrictions.

According to the perdurantist approach, defining a class restriction as temporal consists of moving that restriction to the class that represents its temporal evolution, namely the Time Slice. It must be noted that, by definition, a temporal restriction must relate two instances of 4Dfluents:TimeSlice, so both domain and range classes as necessarily to be time slices. The Assembly Rule *assembleRestriction*/5 described by the Algorithm 14

rule *assembleClass*(C1,C2)
if *(C2 == 4Dfluents:TimeSlice)* **then**
  do : TimeSlice = `createConceptIdentifier`(C1,"_TimeSlice");
  do : `createClass`(TimeSlice);
  do : `addSubClass`(TimeSlice,C2);
  do : `addRestriction`(C1,4Dfluents:hasTimeSlice,"SomeValuesFrom",TimeSlice, "SuperClass");
  do : `addRestriction`(TimeSlice,4Dfluents:tsTimeSliceOf,"SomeValuesFrom",C1, "SuperClass");
  do : `addRestriction`(TimeSlice,4Dfluents:tsTimeInterval,"SomeValuesFrom", 4Dfluents:Interval,"SuperClass");
end

Algorithm 13: Define class as temporal (perdurantist approach)

consists of moving a restriction over an Object Property from the class to its temporal manifestation while ensuring the domain and range condition by defining the range class also as temporal.

rule *assembleRestriction*(C1,P,Q,C2,RT)
if *(isObjectProperty(P))* **then**
  do : TimeSliceC1 = `createConceptIdentifier`(C1,"_TimeSlice");
  do : `createClass`(TimeSliceC1);
  do : `addSubClass`(TimeSliceC1,4Dfluents:TimeSlice);
  do : `addRestriction`(TimeSliceC1,4Dfluents:tsTimeSliceOf,"SomeValuesFrom",C1, "SuperClass");
  do : `addRestriction`(TimeSliceC1,4Dfluents:tsTimeInterval,"SomeValuesFrom", 4Dfluents:Interval,"SuperClass");
  do : `replaceRestriction`(C1,P,Q,C2,4Dfluents:hasTimeSlice,Q,TimeSliceC1,RT);
  do : TimeSliceC2 = `createConceptIdentifier`(C2,"_TimeSlice");
  do : `createClass`(TimeSliceC2);
  do : `addSubClass`(TimeSliceC2,4Dfluents:TimeSlice);
  do : `addRestriction`(TimeSliceC2,4Dfluents:tsTimeSliceOf,"SomeValuesFrom",C2, "SuperClass");
  do : `addRestriction`(TimeSliceC2,4Dfluents:tsTimeInterval,"SomeValuesFrom", 4Dfluents:Interval,"SuperClass");
  do : `addRestriction`(TimeSliceC1,P,Q,TimeSliceC2,RT);
end

Algorithm 14: Define class as temporal (perdurantist approach)

Besides temporal and spatial ontologies, it is possible that the FONTE method can be used to engineer other transversal knowledge categories, such user rights, or to assist the user modeling some notions usually hard to represent using OWL ontologies such different types of Collections (*e.g.*, bags, sets, lists).

5.3 Using FONTE for other ontology operations

Currently there are large amounts of ontologies, of many different types, from which new challenges and opportunities arises for the ontology engineering, such as the reuse

of ontologies and the integration of knowledge from different ontologies of the same or different domains. There are several methods and approaches to tackle the problems related to ontology integration [Pinto and Reiter, 1993], such as: merging, integration or mapping.

Ontology Merging – The objective of the merging technique is to merge two or more ontologies about same knowledge domains, creating a new ontology that represents the unification of these ontologies. This technique is also referred as fusion. The main goal of merging is to overcome the differences between the different models of the same domain (see Figure 5.1).

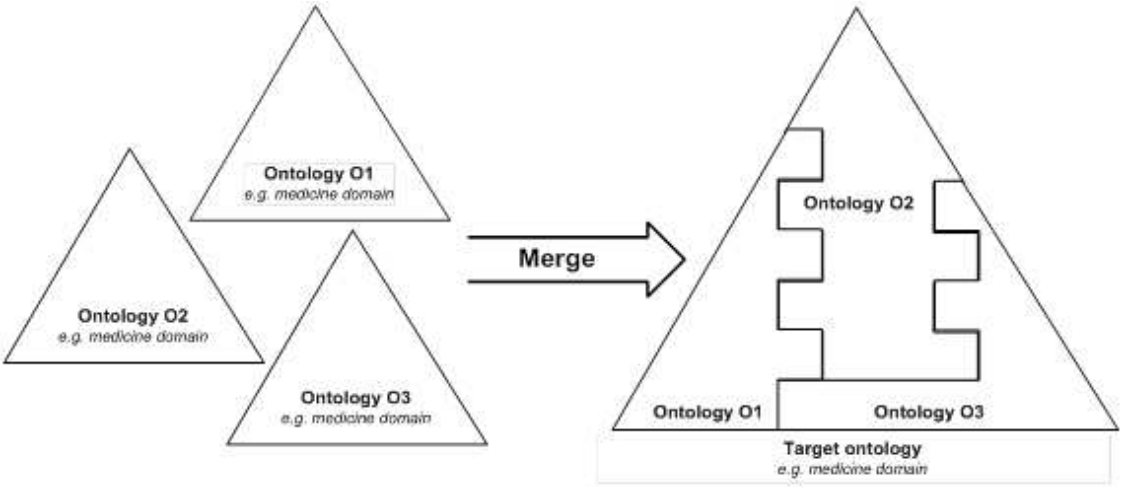


Figure 5.1: Ontology Merging

The optimal result from the application of the merging technique should be an ontology that could reflect the common understanding and consensus for the different communities about this domain. The most effective methods of merging are based in analysis and determination of similarities between ontologies, by comparing classes, instances and taxonomies of both ontologies. These techniques are iterative and the process is repeated for every set of ontologies that is pretended to merge.

Ontology Integration – Ontology integration consists on the construction of an ontology by reusing a part, or the whole, of other ontologies (see Figure 5.2). This process is particularly important since the number of ontologies available in public domain has increased significantly.

In the integration process, there are typically considered operations of inclusion, refinement and specialization of the ontologies to reuse. In the integration context there

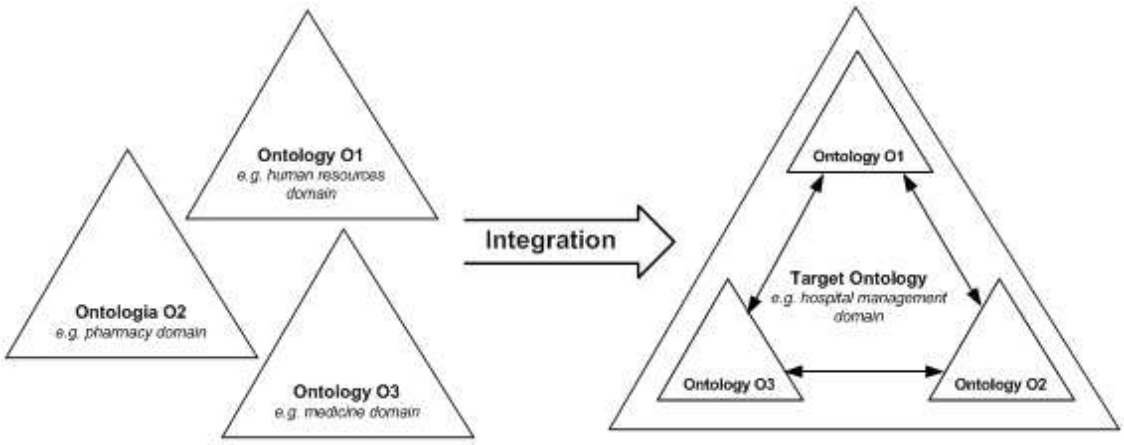


Figure 5.2: Ontology Integration

are references to the reuse of ontologies of same or different domains. Notice that, in the first case, there are no significant differences relatively to merging. Figure 5.2 illustrate an example of ontologies reuse in pharmacy, human resources and medicine domains, with the purpose of create a new ontology, designated hospital management.

Ontology Mapping – The ontology mapping main goal is to determine and establish, either manually or automatically, the rules that allow relating two or more ontologies. The mappings established between concepts and properties are usually referred as semantic bridges.

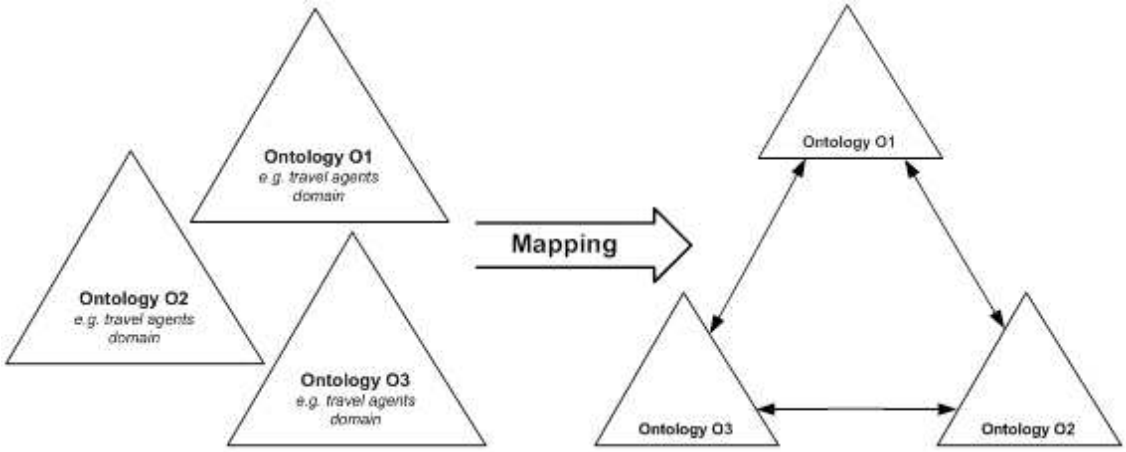


Figure 5.3: Ontology Mapping

The techniques used for mapping share a common approach with the ones used for merging and integration because both need to determine similarity relations. However the mapping process does not produce a new ontology keeping the participant ontologies separated (see Figure 5.3).

So far FONTE was presented as an method to engineer temporal and spatial aspects in domain OWL ontologies through a so called assembly process. Since the FONTE assembly process is propelled by a set of Assembly Rules that can be easily modified to accommodate different user needs it could be argued that FONTE can be configured to assist the user on other operations that involve combining different ontologies.

For instance, an Assembly Rule can be configure to perform a semantic analysis methods to search for similar concepts between two ontologies and then apply a specific set of operations to merge them.

The use of FONTE to perform such ontology operations should be the target of future research.

Chapter 6

Conclusions

The work described in this document was initiated under the FONTE Project created by Jorge Santos [Santos, 2008] under his Ph.D. and pursued by Luís Braga (author) within his internship in order to get his graduation. FONTE main objective was to develop a strategy to integrate temporal representation and reasoning capabilities in intelligent systems by using F-Logic ontologies. Some of the most important characteristics of FONTE is the ability to make suggestions about which tasks must be performed and to ensure the ontology consistency. These efforts were then integrated as part of the WorldSearch Project[1] with the objective of represent and reason about temporal and spatial knowledge in the context of the Semantic Web by using OWL ontologies.

This work includes the study of aspects related to the representation and reasoning of temporal and spatial dimensions in intelligent systems. In particular the different theories of spatial and temporal representation were studied, namely the endurantist and perdurantist approaches, as well as the main aspects related to the fields in question, including the form of representation of its basic entities, the topology, direction, density and temporal granularities and mereotopology, orientation and distance in the case of spatial dimension. Additionally the most important algebras for reason about time and space were investigated, namely the Vilain and Kautz's Point Algebra [Vilain and Kautz, 1986] and Allen's Interval Algebra [Allen, 1983] concerning the temporal dimension; and RCC8 and 2D projection-based method regarding spatial mereotopology and orientation. This study resulted in the section 2.1 of the State of the Art. The elaboration of the State of the Art also consisted in the study of the current status of the Semantic Web and its inherently associated technologies for knowledge representation, namely the RDF, RDFS

[1] `http://www.microsoft.com/portugal/mldc/worldsearch/en/`

and OWL. In the section 2.2 of the State of the Art is explained what is the Semantic Web, why it is so important in the context of the information systems and how it has evolved.

One peculiarity of OWL, the standard for knowledge representation within the Semantic Web, is the lack of formal mechanisms to represent and reason about time and space, making it poorly suited to represent and reason about the temporal and spatial evolution of knowledge. Regarding this problem it was conducted a study about the representation and reasoning of temporal and spatial knowledge in OWL ontologies and how to include those notions in OWL domain ontologies in order to get domain ontologies with the capability to represent and reason about the temporal and spatial evolution of knowledge. Several approaches to include the temporal and spatial dimensions in OWL ontologies were analyzed and presented in section 2.3, namely the extension of Description Logic with temporal and spatial operators, the extension of OWL formal schema, ontology versioning techniques and user defined models. The advantages and disadvantages of each approach were discussed and it was concluded that, given the current state of the technology, the best option is to follow the user defined models approach.

An approach to engineering temporal and spatial notions in OWL domain ontologies was developed and presented in the chapter 3. This approach explores the notion of Ontology Design Pattern and Ontology Modularization and Reuse, since their use results in a better, more comprehensive and best suited ontology for sharing knowledge. It was showed how temporal and spatial notions can be engineered in OWL domain ontologies, according to the endurantist point of view. In particular, it was showed how to temporally and spatially characterize classes, relations and properties while ensuring the ontology consistency. Considering the temporal and spatial dimensions has a huge impact on how the semantics of OWL model should be interpreted. This work shown how some OWL notions such class and property disjunction, and transitive, functional, inverse and symmetric properties can be interpreted and modeled in order to ensure the achievement of the correct reasoning results when considering the temporal and spatial dimensions.

In order to support the ontology engineer in the engineering of temporal and spatial aspects in OWL domain ontologies the FONTE method was re-engineered and presented in chapter 4:

- The FONTE process and architecture were revised to adapt the OWL needs;
- The concept of ODP was explicitly adopted and it is now possible for the user to semi-automatically apply patterns over ontology classes, properties and restrictions;
- It was developed an expressive Assembly Rule Language which allows the possibility to easily and quickly codify different ODPs in order to support different user needs.
- It were developed different strategies to generate proposals. This strategies includes structural and semantic analysis of the participant ontologies and considers the OWL semantic model;
- It was developed a proposals list consistency mechanism that ensures that all new and active FONTE proposals are valid.
- A new graphic tool supporting the FONTE method was designed from scratch. This tools consists of a plug-in for Protégé, one of the most used ontology editors.

In this document it was also showed how FONTE can be used to engineer temporal and spatial aspects in OWL domain ontologies by running an use case example. This example consisted in the engineering temporal and spatial dimensions modeled in an upper ontology (the COSMO ontology) with an domain ontology about football.

The results obtained in the case study was presented and discussed in the chapter 5. The possibility of considering different ontologies and theories about time and space in the assembly process are discussed in this chapter. More specifically two Assembly Rules the 4D-fluentsDEN ontology, which modeled according to the perdurantist view, are presented. These rules allows the temporal characterization of classes and restrictions.

Considering the FONTE adaptability it was also discussed the possibility of using FONTE to perform ontology combining operations such as ontology merging, integration and mapping.

6.1 Limitations

Both the engineering approach presented in chapter 3 and the FONTE method and tool presented in chapter 4 have some limitations.

Considering the temporal and spatial dimensions involves a serious tradeoff between expressivity and decidability. The temporal and spatial engineering of OWL domain ontolo-

gies result in a more expressive but yet more complex ontology, which can seem confusing. Since currently there is no formal and standard approach to so, the commonly followed solutions are based on creating complex user defined models. As was shown, the correct modeling of some OWL characteristics (*e.g.*, disjoint classes or transitive properties) is not straightforward and maintain such ontology may be a difficult task. In particular, reasoning about the temporal evolution of concepts involves the creation of new individuals at runtime, which is problematic since it affects the ontology decidability. The use of SWRL rules helps mitigate the lack of expressiveness of OWL to perform temporal and spatial reasoning. Additionally, the consideration of non standard reasoners can help to overcome the problem of creating individuals at runtime.

Currently, the FONTE tool is only able to perform changes at the TBox level. This means that it does not support the semi-automatic manipulation of any kind of semantic web rules (RBox), which can be problematic when is necessary to ensure some domain business rules such ensuring the temporal disjunction in role playing. Also, additional changes are needed to be performed at the ABox level, since the TBox structural modifications have impact on how the ontology individuals are represented. At the moment FONTE is not capable of assist the user in these kind of modifications.

Some of the presented approaches for Assembly Proposals generation are not yet supported by FONTE namely the Analysis of the involved Data Types and the Lexical Analysis of the involved Property, since they need further analysis and have not been implemented yet.

6.2 Future Work

It seems reasonable to affirm that in all the research, there is room for improvement. In this section some of possible guidelines of future work are pointed out:

Support for RBox manipulation – As previously stated, FONTE tool is not able to realize modifications at the RBox level, which are important to improve the temporal and spatial representing and reasoning capabilities. During the development of this work RIF as become a standard recommended by W3C. The support for RBox manipulation, in particular considering RIF, must be a priority;

Support for ABox manipulation – The development of methods for manipulat-

ing the ontology individuals according to the modifications performed over the classes, properties and restrictions, should be considered in future improvements, since the modifications at the TBox level as direct impact on how the ontology individuals must be represented;

Improve the proposal generation mechanism – In order to provide a better user support the proposal generation mechanisms should be improved, in particular with the implementation of some strategies such Analysis of the involved Data Types and the Lexical Analysis of the involved Property. Also, it would be useful to develop a mechanism that was able to explain why each proposal was generated;

Consideration of other case studies – FONTE was applied on the temporal and spatial engineering of different domains, according to different theories such the endurantist and the perdurantist with significant sucess. However, further studies may be conducted, in particular concerning the perdurantist aproach;

Use of FONTE for other ontology operations – Due to its adaptability nature it is expectable that FONTE can be applied to assist the user on other ontology operations such ontology merging, integration and mapping. Further tests may be conducted to prove this point.

Bibliography

[Allen, 1983] Allen, J. [1983]. Maintaining knowledge about temporal intervals, *Communication ACM* **26**(11): 832–843.

[Artale and Franconi, 2001] Artale, A. and Franconi, E. [2001]. A survey of temporal extensions of description logics, *Annals of Mathematics and Artificial Intelligence* **30**: 171–210.

[Balbiani et al., 1998] Balbiani, P., Condotta, J.-F. and del Cerro, L. F. [1998]. A model for reasoning about bidimensional temporal relations.

[Batsakis and Petrakis, 2010] Batsakis, S. and Petrakis, E. G. M. [2010]. Sowl: spatio-temporal representation, reasoning and querying over the semantic web, *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, ACM, New York, NY, USA, pp. 15:1–15:9.

[Batsakis and Petrakis, 2011] Batsakis, S. and Petrakis, E. G. M. [2011]. Sowl: a framework for handling spatio-temporal information in owl 2.0, *Proceedings of the 5th international conference on Rule-based reasoning, programming, and applications*, RuleML'2011, Springer-Verlag, Berlin, Heidelberg, pp. 242–249.

[Berners-Lee, 2000] Berners-Lee, T. [2000]. Semantic web - xml2000, `http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`. Accessed at 25 September 2013.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J. and Lassila, O. [2001]. The semantic web, *Scientific American* .

[Bezerra et al., 2009] Bezerra, C., Freitas, F., Euzenat, J. and Antoine, J. [2009]. An approach for ontology modularization, *In COLIBRI: Colóquio em Informática: Brasil / INRIA, Cooperações, Avanços e Desafios*, Brasil.

[Bittner et al., 2004] Bittner, T., Donnelly, M. and Smith, B. [2004]. Endurants and perdurants in directly depicting ontologies, *AI Communications, IOS Press* **13**(4): 247–258.

[C. Welty and Makarios, 2006] C. Welty, R. F. and Makarios, S. [2006]. A reusable ontology for fluents in OWL, *In Proceedings of FOIS*, pp. 226–236.

[Carroll et al., 2005] Carroll, J. J., Bizer, C., Hayes, P. and Stickler, P. [2005]. Named graphs, provenance and trust, *Proceedings of the 14th international conference on World Wide Web*, ACM Press, New York, New York, USA, p. 613.

[Clementini et al., 1997] Clementini, E., Felice, P. D. and Hernández, D. [1997]. Qualitative representation of positional information, *Artificial intelligence* **95**: 317–356.

[d'Aquin et al., 2009] d'Aquin, M., Schlicht, A., Stuckenschmidt, H. and Sabou, M. [2009]. Criteria and evaluation for ontology modularization techniques, *in* H. Stuckenschmidt, C. Parent and S. Spaccapietra (eds), *Modular Ontologies*, Vol. 5445 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 67–89.

[Doran, 2006] Doran, P. [2006]. Ontology reuse via ontology modularisation, *In Proceedings of KnowledgeWeb PhD*, Budva, Montenegro.

[Fisher et al., 2005] Fisher, M., Gabbay, D. and Vila, L. (eds) [2005]. *Handbook of Temporal Reasoning in Artificial Intelligence*, Vol. 1 of *Foundations of Artificial Intelligence Series*, Elsevier Science & Technology Books.

[Frank, 1991] Frank, A. U. [1991]. Qualitative spatial reasoning with cardinal directions, *ÖGAI*, pp. 157–167.

[Freksa, 1992] Freksa, C. [1992]. *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, Springer Berlin Heidelberg, chapter Using Orientation Information for Qualitative Spatial Reasoning, pp. 162–178.

[Gangemi, n.d.] Gangemi, A. [n.d.]. Nary Participation, `http://ontologydesignpatterns.org/wiki/Submissions:Nary_Participation`.

[Gangemi and Presutti, 2009] Gangemi, A. and Presutti, V. [2009]. *Handbook on Ontologies*, 2nd edn, Springer, chapter Ontology Design Patterns.

[Gerevini, 1997] Gerevini, A. [1997]. Reasoning about time and actions in artificial intelligence: Major issues, *in* O.Stock (ed.), *Spatial and Temporal Reasoning*, Kluwer Academic Públishers, pp. 43–70.

[Goyal and Egenhofer, in press] Goyal, R. and Egenhofer, M. [in press]. Cardinal directions between extended spatial objects, *IEEE Transactions on Knowledge and Data Engineering*, IEEE.

[Gruber, 1993] Gruber, T. [1993]. Towards Principles for the Design of Ontologies Used for Knowledge Sharing, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer, pp. 93–104.

[Guarino and Welty, 2000] Guarino, N. and Welty, C. [2000]. A formal ontology of properties, *Knowledge Acquisition, Modeling and Management*, pp. 97–112.

[Gutierrez et al., 2007] Gutierrez, C., Hurtado, C. A. and Vaisman, A. [2007]. Introducing time into rdf, *IEEE Trans. on Knowl. and Data Eng.* **19**: 207–218.

[Hahmann and Grüninger, 2010] Hahmann, T. and Grüninger, M. [2010]. *Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions*, IGI Publishing, chapter Region-based Theories of Space: Mereotopology and Beyond, pp. 1–62.

[Hayes et al., 2002] Hayes, P., Lehmann, F. and Welty, C. [2002]. *Endurantism and Perdurantism: An Ongoing Debate*. Published:
urlhttp://ontology.teknowledge.com:8080/rsigma/dialog-3d-4d.html.

[Hogenboom et al., 2010] Hogenboom, F., Frasincar, F. and Kaymak, U. [2010]. A review of approaches for representing rcc8 in owl., *in* S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal and C.-C. Hung (eds), *SAC*, ACM, pp. 1444–1445.

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B. and Dean, M. [2004]. Swrl: A semantic web rule language combining owl and ruleml, `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`. Accessed at 23 September 2013.

[Isli and Moratz, 1999] Isli, A. and Moratz, R. [1999]. Qualitative spatial representation and reasoning: algebraic models for relative position, *Technical report*, Universität Hamburg, FB Informatik.

[Klein and Fensel, 2001] Klein, M. and Fensel, D. [2001]. Ontology versioning on the semantic web, *Proc. 1st Int. Semantic Web Working Symp.*, Stanford University, CA, USA, pp. 75–91.

[Krieger, 2008] Krieger, H.-U. [2008]. Where temporal description logics fail: Representing temporally-changing relationships, *Proceedings of the 31st annual German conference on Advances in Artificial Intelligence*, KI '08, Springer-Verlag, Berlin, Heidelberg, pp. 249–257.

[Lutz et al., 2008] Lutz, C., Wolter, F. and Zakharyaschev, M. [2008]. Temporal description logics: A survey, *Proceedings of the 2008 15th International Symposium on Temporal Representation and Reasoning*, IEEE Computer Society, Washington, DC, USA, pp. 3–14.

[Ma and Wang, 2012] Ma, Z. M. and Wang, X. [2012]. Rule interchange in the semantic web., *J. Inf. Sci. Eng.* **28**(2): 393–406.

[Milea et al., 2011] Milea, V., Frasincar, F. and Kaymak, U. [2011]. tOWL: A Temporal Web Ontology Language, *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions* **PP**(99): 1–14.

[Nebel, 2007] Nebel, J. R. B. [2007]. Qualitative spatial reasoning using constraint calculi., *in* M. Aiello, I. Pratt-Hartmann and J. van Benthem (eds), *Handbook of Spatial Logics*, Springer, pp. 161–215.

[Noy and Rector, 2006] Noy, N. and Rector, A. [2006]. Defining N-ary Relations on the Semantic Web, `http://www.w3.org/TR/swbp-n-aryRelations/`.

[O'Connor and Das, 2011] O'Connor, M. and Das, A. [2011]. A method for representing and querying temporal information in owl, *in* S. Verlag (ed.), *In Proc. of Biomedical Engineering Systems and Technologies (Selected Papers)*, number 127, pp. 97–110.

[Piaget and Inhelder, 1972] Piaget, J. and Inhelder, B. [1972]. La représentation de l'espace chez l'enfant.

[Pinto and Reiter, 1993] Pinto, J. and Reiter, R. [1993]. Temporal reasoning in logic programming: A case for the Situation Calculus, *International Conference on Logic Programming*, pp. 203–221.

[Randell et al., 1992] Randell, D. A., Cui, Z. and Cohn, A. G. [1992]. A spatial logic based on regions and connection, *Proc. of 3rd International Conference on Knowledge Representation and Reasoning*.

[Santos, 2008] Santos, J. [2008]. *Temporal Knowledge Verification and Assembling in Intelligent Systems (portuguese)*, PhD thesis, Universidade de Trás-os-Montes e Alto Douro.

[Santos et al., 2011] Santos, J., Braga, L. and Cohn, A. G. [2011]. Fonte: A protégé plug-in for engineering complex ontologies, *in* W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, J. Filipe and J. Cordeiro (eds), *Enterprise Information Systems*, Vol. 73 of *Lecture Notes in Business Information Processing*, Springer Berlin Heidelberg, pp. 222–236.

[Santos and Staab, 2003a] Santos, J. and Staab, S. [2003a]. Engineering a complex ontology with time, *18th International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco/Mexico, pp. 1406–1407.

[Santos and Staab, 2003b] Santos, J. and Staab, S. [2003b]. FONTE - Factorizing ONTology Engineering complexity, *The Second International Conference on Knowledge Capture (K-Cap'03)*, Florida/USA, pp. 146–153.

[Shadbolt et al., 2006] Shadbolt, N., Berners-Lee, T. and Hall, W. [2006]. The semantic web revisited, *IEEE Intelligent Systems* **21**(3): 96–101.

[Sirin and Parsia, 2007] Sirin, E. and Parsia, B. [2007]. Sparql-dl: Sparql query for owl-dl, *In 3rd OWL Experiences and Directions Workshop (OWLED-2007.*

[Sowa, 2006] Sowa, J. [2006]. Semantic networks, `http://www.jfsowa.com/pubs/semnet.htm`. Accessed at 4 October 2013.

[Stock, 1997] Stock, O. [1997]. *Spatial and Temporal Reasoning*, Kluwer Academic Publishers, Norwell, MA, USA.

[Stuckenschmidt, 2003] Stuckenschmidt, H. [2003]. Modularization of ontologies - wonderweb: Ontology infrastructure for the semantic web.
URL: *http://citeseer.ist.psu.edu/stuckenschmidt01modularization.html*

[Studer et al., 2004] Studer, R., Decker, S., Fensel, D. and Staab, S. [2004]. *Knowledge Engineering and Agent Technology*, Vol. 52, IOS Press, chapter Situation and Perspective of Knowledge Engineering.

[van Harmelen et al., 2007] van Harmelen, F., Lifschitz, V. and Porter, B. (eds) [2007]. *Handbook of Knowledge Representation (Foundations of Artificial Intelligence)*, Elsevier Science.

[Vieu, 1997] Vieu, L. [1997]. *Spatial and Temporal Reasoning*, O.Stock, editor, Kluwer, chapter Spatial Representation and Reasoning in Artificial Intelligence, pp. 5–41.

[Vila and Schwalb, 1996] Vila, L. and Schwalb, E. [1996]. A theory of time and temporal incidence based on instants and periods, *Proc.International Workshop on Temporal Representation and Reasoning* pp. 21–28.

[Vilain and Kautz, 1986] Vilain, M. and Kautz, H. [1986]. Constraint propagation algorithms for temporal reasoning, *Proc. of AAAI-86*, Philadelphia, PA, pp. 377–382.

[Vilain et al., 1989] Vilain, M., Kautz, H. and Beek, P. [1989]. Constraint propagation algorithms: a revised report, *Readings in Qualitative Reasoning about Physical Systems* .

[Völkel et al., 2006] Völkel, M., Kruk, S. R., Zhdanova, A. V., Stevens, R., Artale, A., Franconi, E. and Tessaris, S. [2006]. Semversion - versioning rdf and ontologies, *Knowledge web deliverable*, University of Karlsruhe.

[Wessel, 2002] Wessel, M. [2002]. On spatial reasoning with description logics, *Proceedings of the International Workshop on Description Logics 2002 (DL2002), number 53 in CEUR-WS*, pp. 156–163.

[Xu et al., 2008] Xu, X.-h., Huang, J.-l., Wan, J. and Jiang, C.-f. [2008]. A method for measuring semantic similarity of concepts in the same ontology, *Proceedings of the 2008 International Multi-symposiums on Computer and Computational Sciences*, IEEE Computer Society, Washington, DC, USA, pp. 207–213.
URL: *http://dl.acm.org/citation.cfm?id=1491268.1493205*