OntoVersionGraph : a change management methodology dedicated to formal ontologies and their user views in a collaborative context

Perrine Pittet

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

Présentée pour obtenir
Le grade de Docteur ès Sciences
Mention Informatique
Par

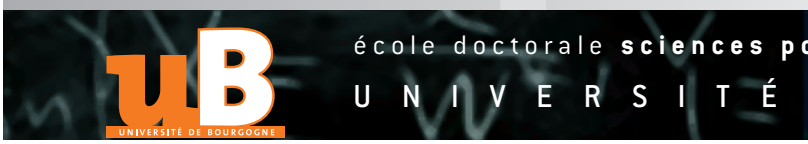
Perrine Pittet

OntoVersionGraph: A Change Management Methodology Dedicated to Formal Ontologies and their User Views in a Collaborative Context: Application to SHOIN (D) Ontologies

Le 11 juillet 2014

SPIM

Thèse de Doctorat



école doctorale **sciences pour l'ingénieur et microtechniques**

U N I V E R S I T É D E B O U R G O G N E

Composition du jury :

Pr. Luciano SERAFINI, Professeur des Universités, DKM, IFB, Trento, **Rapporteur**

Pr. Nuno SILVA, Professeur des Universités, DEI & GECAD, ISEP/IPP, Porto, **Rapporteur**

M. Rim DJEDIDI, Maître de Conférence, LIM&BIO, Université de Paris 13, Bobigny, **Examinateur**

Pr. Wassim JAZIRI, Professeur des Universités, MIRACLISIMS, Sfax, **Examinateur**

M. Christophe CRUZ, Maître de Conférences HDR, Le2i, UB, Dijon, **Co-Directeur de Thèse**

Pr. Christophe NICOLLE, Professeur des Universités, Le2i, UB, Dijon, **Directeur de Thèse**

Doctorat préparé au sein de l'équipe Systèmes d'Information et Systèmes d'Images, dans l'équipe projet CHECKSEM, du Laboratoire Électronique, Informatique et Image, UMR CNRS 6306, à l'Université de Bourgogne.

Résumé en Français

Le monde change au fil du temps. Cela impacte la connaissance de chaque sous-domaine qu'il contient. Par conséquent, les systèmes décrivant la connaissance d'un certain sous-domaine du monde devraient être en mesure de tenir compte de ses changements afin de maintenir la représentation de ses connaissances à jour. L'ontologie formelle est l'un d'eux: elle représente explicitement et formellement la connaissance d'un domaine sous toutes ses formes et modes d'existence. Développée en collaboration par des experts du domaine, elle permet aux usagers d'un domaine de se comprendre en partageant la même terminologie sur ce domaine, malgré les différentes hypothèses que ceux-ci portent sur sa conceptualisation. Toutefois, à cause de son exhaustivité, la complexité de sa conceptualisation peut parfois rendre inaccessible la connaissance du domaine à ses usagers. Aussi les usagers du domaine n'ont souvent besoin d'accéder qu'à un sous-ensemble de celle-ci. Des sous-portions de l'ontologie ou vues définies selon leurs hypothèses devraient donc être produites pour leur fournir une vue personnalisée et compréhensible de la connaissance décrite par celle-ci. Considérant une telle ontologie, les changements du monde peuvent avoir un impact à la fois sur la connaissance du domaine mais aussi sur les hypothèses de ses usagers. Cependant, leur application peut rendre incohérente la connaissance représentée par l'ontologie et par ses vues usagers. Aussi, une ontologie incohérente n'est plus utilisable par la communauté. La gestion du changement de l'ontologie doit assurer le maintien de sa cohérence afin que les usagers du domaine puissent continuer à coopérer et à se comprendre. Comme une ontologie formelle requiert un développement collaboratif pour prendre en compte chaque hypothèse usager, il en va de sa gestion du changement. Le problème est donc le suivant : comment mener une gestion du changement collaborative dédiée aux ontologies formelles prenant en compte les changements survenus dans le domaine ainsi que les nouvelles hypothèses usager ? Les travaux de recherche existant au sein du Web sémantique ne permettent actuellement pas une telle gestion du changement ontologique. C'est pourquoi la contribution de ma thèse consiste à concevoir une approche de gestion du changement dédiée aux ontologies formelles, permettant : la spécification de vues usagers sur une ontologie sur la base de nouvelles hypothèses usagers, l'évolution collaborative et cohérente de l'ontologie et de ses vues, et la gestion de l'impact des changements entre celles-ci.

Abstract

The world changes over time, impacting the knowledge of every subdomain it contains. Therefore systems describing the knowledge of a certain domain should be able to consider changes occurred to keep its knowledge representation up-to-date. Formal ontologies are one of them: they explicitly and formally represent the knowledge of a domain in all its forms and modes of existence. Collaboratively developed, a formal ontology allows the domain users to understand each other by sharing the same terminology despite the different assumptions they have on the domain conceptualization. However, due to its completeness, the complexity of its conceptualization can sometimes make the domain knowledge inaccessible for its users. Also domain users often need to access only a subset of this knowledge. Light sub-portions of the ontology or views defined by their assumptions should then be produced to provide domain users a personalized and comprehensive view of the domain knowledge described by the ontology. Considering such ontology, the world changes may impact both domain knowledge and users assumptions and their application can turn inconsistent the knowledge represented by the ontology and its views. However, an inconsistent ontology is no longer usable by the domain community. Ontology change management should ensure maintaining its consistency, so that its users can continue to cooperate and understand each other. Also, a formal ontology requires a collaborative development to take into account every domain user assumption, so does its change management. The problem is the following: how to enable collaborative ontology change management considering changes in the domain and new domain users assumptions? Until now, existing research in the Semantic Web does not allow such change management for formal ontologies. The contribution of my thesis therefore consists in designing an approach to ontology change management dedicated to formal ontologies enabling: the specification of ontology views based on new domain users assumptions, the consistent collaborative evolution of the ontology and its views, and the change impact management between them.

Remerciements

Mes remerciements vont tout d'abord à mon directeur de thèse, Pr. Christophe Nicolle, et à mon co-directeur, M. Christophe Cruz, sans qui rien de tout cela n'aurait été possible. Ils ont su me transmettre leur expérience et leur expertise, ainsi que leur motivation tout au long de cette aventure.

Je tiens également à remercier les membres de mon jury qui m'ont fait l'immense honneur d'être tous présents à ma soutenance: le président du jury Pr. Wassim Jaziri, les rapporteurs Pr. Luciano Serafini et Pr. Nuno Silva, ainsi que M. Rim Djedidi. Ils ont été pour moi une source d'inspiration essentielle dans mes recherches et ont rendu inoubliable cette journée.

Un grand merci également à tous mes collègues de l'équipe Checksem, du laboratoire Le2i, de l'IUT de Dijon et de l'UFR Sciences et Techniques, ainsi qu'à toutes les personnes que j'ai pu rencontrer et qui ont ponctué ces années de doctorat et d'enseignement de discussions très enrichissantes.

Je n'oublie pas non plus mes amis, ma famille, ma belle-famille, et tous ceux qui, par leur soutien quotidien, m'ont donné la force suffisante pour aller jusqu'au bout de la rédaction. Cette thèse est aussi la leur.

Enfin je remercie Alex, pour avoir été à mes côtés, même dans les moments de doute, pendant ces 4 années de doctorat, et pour l'être, encore aujourd'hui.

Table of Contents

Chapter 1 19
1 Introduction 21
1.1 Background: The Semantic Web 21
1.2 General Research Context: Formal Ontology 22
1.3 Specific Research Context: Knowledge Evolution 23
1.4 Problem: Formal Ontology Change Management 23
1.5 Global Approach 24
1.6 Contributions 24
1.7 Limitations 24
2 Document Organization 26
2.1 Chapter 2: General Research Field – Formal Ontology 26
2.2 Chapter 3: Specific Research Fields – OCM & Ontology Views 26
2.3 Chapter 4: $\mathcal{SHOIN}(\mathcal{D})$ Collaborative OCM: Methodology and Model 26
2.4 Chapter 5: $\mathcal{SHOIND}$ Structural and Logical Consistency Preventive Resolution 26
2.5 Chapter 6: $\mathcal{SHOIND}$ Ontology Views Specification & Sub-Ontology Extraction 27
2.6 Chapter 7: Change Impact Management of $\mathcal{SHOIND}$ Ontology & Views 27
2.7 Chapter 8: Prototype Implementation 27
2.8 Chapter 9: Conclusions and Future Works 28
Chapter 2 33
1 Ontologies and Formal Ontology 35
1.1 Ontology in Computer Science 35
1.2 Ontology Definitions 36
1.3 Formal Ontology in Applied Ontology Research Field 39
1.3.1 Ontology: Study of Content 39
1.3.2 Formal Ontology Building Process 41
1.4 Conclusion 42
2 Formal Ontology Characteristics 43
2.1 Formal Ontology Composition 43
2.2 Formal Ontology and Interpretation 43
2.3 Formal Ontology and Inference 44
2.3.1 Terminological Level Inference 44
2.3.2 Assertional Level Inference 46
2.4 Formal Ontology Quality 47
2.5 Formal Ontology and Language 48
2.5.1 A Formal Language 48
2.5.2 Formal Language Expressivity 48
2.5.3 Semantic Web Representation Languages 49
2.5.4 Description Logics vs. OWL for Formal Ontology Representation 50
2.6 Discussion 52
3 Conclusion 53
Chapter 3 59
1 Ontology Change Management 60
1.1 Ontological Changes 60

1.1.1 What is an Ontological Change? 60
1.1.2 Ontological Change Types Classification 61
1.2 Ontology Evolution 63
1.2.1 Ontology Evolution Definitions 63
1.2.2 Ontology Consistency Maintenance 64
1.3 Ontology Versioning: 66
1.3.1 Ontology Versioning : Definitions 66
1.3.2 Representing and Tracing Changes for Versioning Purposes 67
1.4 Ontology Change Management 68
1.4.1 Klein and Stojanovic's Methodologies 68
1.4.2 Related Works 69
1.5 Discussion 70
2 Ontology Views 73
2.1 Ontology Views in the Literature 73
2.2 View Specifications Approaches 74
2.2.1 Query Language Approaches 74
2.2.2 Manual, Structural and Logical Approaches 76
2.2.3 Other Approaches: Rules, Pipes and NamedGraphs 76
2.3 Discussion 77
3 Conclusion: Deadlocks and Approach of my Proposal 80
Chapter 4 89
1 OntoVersionGraph: A $\mathcal{SHOIN}(\mathcal{D})$ Ontology Collaborative Change Management Methodology 91
1.1 Change Detection Phase 91
1.2 Change Modelling Phase 93
1.3 Change Semantics Phase 93
1.4 Change Implementation Phase 94
1.5 Change Propagation Phase 94
1.6 Change Validation Phase 95
1.7 Discussion 96
2 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model 97
2.1 Preliminary Notions 97
2.2 Model definition of the structure for a $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ Ontology 98
2.3 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Example 99
3 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Change Modelling 104
3.1 Change Definitions 104
3.2 $\mathcal{SHOIN}(\mathcal{D})$ Basic Changes Representation 105
3.3 $\mathcal{SHOIN}(\mathcal{D})$ Composed Changes Representation 106
4 Conclusion 108
Chapter 5 113
1 Introduction on $\mathcal{SHOIN}(\mathcal{D})$ Ontology Consistency 115
2 Structural Consistency Preventive Approach 117
2.1 $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraints 117
2.2 Structural Consistency Resolution Example 118
2.3 Discussion 119
3 Logical Consistency Preventive Approach 121
3.1 $\mathcal{SHOIN}(\mathcal{D})$ Inconsistency 121

3.2 $\mathcal{SHOIN}(\mathcal{D})$ Logical Inconsistency Preventive Approach 123
3.2.1 Basic $\mathcal{SHOIN}(\mathcal{D})$ Change Unsatisfiability Constraints 123
3.2.2 Composed $\mathcal{SHOIN}(\mathcal{D})$ Change Unsatisfiability Constraints 125
3.2.3 Logical Consistency Preventive Approach Process 126
3.3 Logical Consistency Preventive Analysis Example 127
3.3.1 Logical Consistency Detection Example 128
3.3.2 Logical Consistency Resolution Example 140
3.4 Discussion 140
4 Conclusion 141
Chapter 6 145
1 $\mathcal{SHOIND}$ View Extraction Approach 148
1.1 Preliminary Definitions 148
1.2 Extraction Structural Constraints 149
1.3 Extraction Levels 151
1.4 Discussion 152
2 View Extraction Formalization 153
2.1 $\mathcal{SHOIN}(\mathcal{D})$ View Extraction 153
2.2 Starter Extraction rules 154
2.3 Complete Extraction rules 154
2.4 Partial extraction rules 159
2.5 Sub-ontology Global Log of Changes Generation Rule 160
2.6 View Extraction Example 161
2.7 View Extraction Result 162
2.8 View and Ontology Global Logs of Changes 167
2.9 Discussion 168
3 Conclusion 170
Chapter 7 175
1 Top-Down Propagation Approach 178
1.1 Ontology Impact Rule 179
1.2 View Impact Rule 179
1.3 View Update Rule 180
1.4 Sub-Ontology Update Rule 180
1.5 Discussion 181
2 Bottom-Up Propagation Approach 182
2.1 $\mathcal{SHOIND}$ Ontology Update Rule 183
2.2 Source Ontology Evolution 183
2.3 Source Ontology Changes Propagation 184
2.4 Discussion 184
3 Change Impact Management Example 185
3.1 Assertional Change Example 185
3.1.1 Introduction to the problem 185
3.1.2 Architect Change Management Process: Round 1 187
3.1.3 Electrician Change Management Process 188
3.1.4 Architect Change Management Process: Round 2 190
3.2 Terminological Change 190
3.3 Discussion 193

4 Conclusion 194
Chapter 8 197
1 OntoVersioning API: Presentation of the Prototype 200
1.1 API Main Features 200
1.1.1 Ontology Change Management Module 200
1.1.2 Ontology View Specification & Sub-Ontology Extraction Module 203
1.1.3 Change Impact Management Module 203
1.2 VersionGraph Content 204
1.3 Other API Modules 206
1.3.1 Project Creation Module 206
1.3.2 Project Modification Module 207
1.3.3 OntoVersioning Project Propagation 208
1.4 Discussion 208
2 Using OWL DL Ontology Change Management in OntoVersioning 209
2.1 Choosing an Operation Verification Mode 209
2.2 Modelling Changes 209
2.3 Implementing and Validating Changes 212
2.4 Discussion 213
3 Specifying Views & Extracting Sub-Ontologies in OntoVersioning 215
3.1 Defining a View 215
3.2 View Extraction 215
3.3 Discussion 216
4 Using Change Impact Management Module in OntoVersioning 220
4.1 Using the Top-Down Propagation Approach 220
4.2 Using the Bottom-Up Propagation Approach 221
4.3 Discussion 222
5 Conclusion and Future Development Perspectives 223
Chapter 9 227
1 Contribution 229
2 Research Perspectives 231

Chapter 1

Introduction

Summary

This chapter constitutes the **introduction** of my thesis. The first section **introduces the subject** by defining the background, the general and specific research contexts, the related problem, the global approach and the contributions and limitations of my proposal. The second section gives a **brief overview of each chapter** organizing this document.

Plan

1 Introduction 21
1.1 Background: The Semantic Web 21
1.2 General Research Context: Formal Ontology 22
1.3 Specific Research Context: Knowledge Evolution 23
1.4 Problem: Formal Ontology Change Management 23
1.5 Global Approach 24
1.6 Contributions 24
1.7 Limitations 24
2 Document Organization 26
2.1 Chapter 2: General Research Field – Formal Ontology 26
2.2 Chapter 3: Specific Research Fields – OCM & Ontology Views 26
2.3 Chapter 4: $\mathcal{SHOIN}(\mathcal{D})$ Collaborative OCM: Methodology and Model 26
2.4 Chapter 5: SHOIND Structural and Logical Consistency Preventive Resolution 26
2.5 Chapter 6: SHOIND Ontology Views Specification & Sub-Ontology Extraction 27
2.6 Chapter 7: Change Impact Management of SHOIND Ontology & Views 27
2.7 Chapter 8: Prototype Implementation 27
2.8 Chapter 9: Conclusions and Future Works 28

My dream for the Web has two parts. In the first, I see the Web becoming a much more powerful means for collaboration among people. In the second, collaborations extend to computers. […] Once the two-part dream is achieved, the Web will be a place where the whim of a human being and the reasoning of a machine will co-exist in an ideal, powerful mixture. (Berners-Lee, 2000)

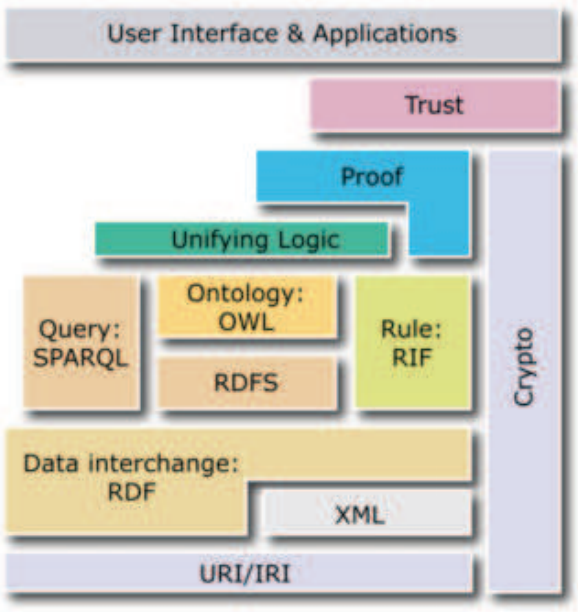
1 Introduction

This section **introduces the subject** by defining the background of my thesis: the Semantic Web. It also briefly presents the general and specific research contexts, the related problem, the global approach and the contributions and limitations of my proposal.

1.1 Background: The Semantic Web

Tim Berners-Lee, the founder of the Web as we know it, outlined the contours of the **Semantic Web** almost 15 years ago. The Semantic Web objective is to help the emergence and sharing of new knowledge from the current Web, to resolve **semantic heterogeneity** between the different systems that use it. Semantic heterogeneity stands for the existence of variations in the way information is specified and structured according to its source, which gives it a **different meaning.** For example, the word "bow" in natural language does not have the same meaning from the point of view of an archer and from the perspective of a violinist. Computer systems require sharing the same meaning when exchanging data to make it explicit and foster human understanding thereof, i.e. **semantic interoperability**. To achieve this **knowledge** must be extracted from this data or be added thereto. For this purpose, the Semantic Web implements the **Web of Data,** which consists in **describing** data, structuring and linking their description over the Internet to access the knowledge they already contain.

Figure 1. Semantic Web Stack



In this context, Tim Berners Lee, has, since 2000, proposed a road map through the **Semantic Web Stack** (Berners Lee, 2000a). The layers of this stack define the different stages of **technological standardizations** required for the implementation of the Web of Data. The lower layers of the stack contain technologies that are well known from the hypertext Web, on which other layers are based. The intermediate layers contain technologies to extract, structure and link the current web knowledge for the Semantic Web dedicated applications. The upper layers contain technologies that enable **the identification, validation and reliability** of such knowledge. A final layer, evolving along with the other layers will produce interfaces between humans and the Semantic Web applications. Today, only technologies of the lower and intermediate layers have been standardized by the **W3C**[1]. Among these standards appear the **knowledge representation languages** RDF[2], RDF-S[3] and

[1] *W3C is the main international standards organization for the World Wide Web. It is responsible for promoting the compatibility of Web technologies: http://www.w3.org/*
[2] *RDF (Resource Description Framework): http://www.w3.org/RDF/*
[3] *RDF (Resource Description Framework): http://www.w3.org/RDF/*

OWL[4]. RDF (Resource Description Framework) is a resource description model used to interconnect Web resources identified through IRI[5]. RDF-S (RDF Schema) is an extension of RDF, which describes these resources by semantically annotating them by means of **concepts**. RDF-S provides the possibility of building hierarchical relationships between these concepts, called **subsumption relations**. It produces **taxonomies,** describing the knowledge shared by the users of a certain domain of the Web. Also other information can be specified in RDF-S which concerns domain and range of a relation. OWL (Web Ontology Language) is an extension of RDF-S producing semantically richer and more precise vocabularies, through multiple **semantic relations** and **logical constraints**. These vocabularies, called **ontologies** have the objective of explicitly specifying and structuring the knowledge of a domain, avoiding ambiguity, so that it can be understood and shared by its whole community. To foster community understanding, the development of an ontology must follow a collaborative and incremental process. Experts of the domain are responsible for the **ontological analysis** thereof. They identify and agree on the **concepts** handled by the community and their structure, thereby producing a **conceptualization** of the domain. Then they determine together the corresponding **terms** that will be used by the community through a **language.** Therefore the community can share the same **terminology** to exchange data. To summarize, an ontology **unifies the meaning of terms within a domain** in **a consensus** with experts of the domain: it allows **semantic integration.**

1.2 General Research Context: Formal Ontology

Two basic scenarios exist for the application of semantic integration. The first is the exchange of data between systems in **a domain** with a **shared terminology** but a **different conceptualization.** The second is the sharing of **the same domain** with **the same terminology** and the same **conceptualization.** According to (Guarino, 2005), ontologies were originally born to solve the semantic heterogeneity problem caused by the first scenario. Experience shows that, in practice, they were mostly used to share data schemes in the second scenario. This had the effect of creating "data silos", where heterogeneous data are stored independently of each other in silos. Data, rather than being unified under a common model, they are then inaccessible to each other.

Several definitions for ontologies coexist in the literature. (Cocchiarella, 1991) proposed the definition of **formal ontology,** supported by (Guarino 1995), which allows taking into account the first scenario: "a systematic, formal and axiomatic development of the logic of all forms and modes of existence." "Systematic development of the logic" means that the ontology is based on a **logical system of knowledge.** An ontology is **"formal",** if this knowledge is **explicitly represented** in a **formal language,** which does **not** tolerate **ambiguity.** Such a formal ontology can be represented by **description logics** and their derivatives including **OWL.** For instance, concepts expressions are the main components of Description logics languages and they are organized in a partial order by the relation type/subtype (Sowa, 1999), as opposed to thesaurii, which are either undefined or defined by assertions in natural language. Also, knowledge is represented in the form of **axioms,** that is to say **obvious and non-contradictory propositions** respecting the logical constraints, hence the term "axiomatic". "In all its forms and modes of existence" means that knowledge must be shared and understood by users of the same domain whatever the **context.**

This formal definition of ontology adds an **explicit and systematic representation** of properties of the model and its elements to the notion of shared conceptualization. This representation, clearly, precisely, and logically describing the knowledge of a domain, allows users to cooperate despite the heterogeneity of their conceptual points of view, i.e. their **assumptions**. This is why a formal ontology is

[4] *OWL (Web Ontology Language): http://www.w3.org/TR/owl-features/*
[5] *IRI (International Resource Identifier): https://www.ietf.org/rfc/rfc3987.txt*

adapted to solve the semantic heterogeneity between systems of the same domain with a shared terminology but a different conceptualization.

1.3 Specific Research Context: Knowledge Evolution

The world is changing over time and knowledge of the world is affected by its changes. This is also the case for knowledge domains that coexist on the Web. **Assumptions** that the community has on this domain may also change. According to different causes, each user of the domain can change his point of view over time. Knowledge described by the ontology must be **updated** so that the community can continue to cooperate and understand each other. In addition, each new user assumption must be taken into account in the **evolution of the ontology.** However, the evolution of an ontology, which consists in **applying changes to the terminology,** can turn the ontology **inconsistent.** Indeed, the application of a change non-compliant with the **syntactic constraints of the language** of representation, for instance if the language constructors are misused, can turn the ontology **structurally inconsistent.** Also, its non-compliance with **logical constraints defined by the axioms** of terminology can create contradictions and make the ontology **logically inconsistent.** An inconsistent ontology is no longer usable by the community, which will no longer be able to access its knowledge or understand each other. Moreover, taking into account new user assumptions can also create contradictions with existing assumptions in the ontology. As for the ontology creation phase, a consensus must be found on how to take into account these new assumptions. Therefore, the evolution of a formal ontology has to be **guided in a formal approach,** i.e. respecting its formal constraints to retrieve a consensus taking into account the new users assumptions.

1.4 Problem: Formal Ontology Change Management

However, a formal ontology describing all the assumptions of the community can quickly become **too complex and inaccessible** to a user of the community and even to a domain expert. Having a complete **understanding of the overall conceptualization** provided by such an ontology is often illusory. Moreover, a user rarely **needs to access the** whole assumptions of the domain. For example, in a collaborative facility management project, if the domain described is a building, an architect user will not need to access the business assumptions of an electrician user. Access to unnecessary knowledge can indeed slow down his task or increase the complexity of the latter. He requires access to a **view** of building knowledge specific to his business concerns. It is therefore essential to **provide** users of a domain **custom access to a portion of the corresponding ontology** based on their own assumptions.

A first issue arises **when a user assumption changes**. This change should be reflected in the ontology but also in each domain user view. It also may imply to reconsider the definition of the view of the ontology, which is provided to the user. In this case, a new portion must be generated from the update definition of the new user assumption. For example, an electrician user being promoted as a heating technician will need to access the building knowledge according to his new business skills. But this new assumption can also cause a need to modify the ontology. For example, if a user or an expert decides to change the knowledge of the building.

A second issue arises in this case. A new evolution of the ontology is required. Furthermore, an expert whose business concerns relate to a subdomain of the ontology will also need a custom access. It will enable him to propose changes to the ontology according to his own assumption without being overwhelmed by the whole ontology complexity. However, the implementation of these changes, even if **logically consistent according to** the expert's assumption, may **generate contradictions in other user views** and, by extension, in the global ontology. For example, an architect expert decides to delete a wall between two rooms of a building. But this wall contains electrical conducts, which do not appear in his

view but are represented in the view of the electrician. Removing this wall will, therefore, turn the building knowledge inconsistent for the electrician and for the overall ontology. It requires other changes in the concerned views to find a consistent ontology. It is, therefore, necessary to **check the consistency of the application of changes in the whole ontology** and integrate a **collaborative approach to solving the consistency.** Indeed, as the evolution of a formal ontology requires consensus among the experts in the field, applying a change, consistent or not, must be validated by other experts. This requires **propagating proposals for change from an expert view of the ontology to the others** so that the other experts can propose potential changes to achieve a consensus.

Until now, existing research in the Semantic Web does not allow a formal ontologies change management as defined in this thesis. The contribution of my thesis, therefore, consists in designing an approach to a **change management** dedicated to formal ontologies, which will allow the specification of views based on domain users assumptions, i.e. their context such as business skills, profiles etc., guiding the coherent collaborative evolution of the ontology and its views by propagating changes from one to the other.

1.5 Global Approach

The global approach of my thesis is articulated in three steps. The first step is the study of the research fields dedicated to formal ontologies, ontology change management (OCM) and ontology views. It aims at identifying existing solutions that could potentially resolve the problem and evaluate them. The second step is based on the results of the study to formalize a methodology for change management dedicated to formal ontologies. The third step is the implementation of this methodology in an API.

1.6 Contributions

The major contribution of this approach lies in the formalization of a **formal OCM methodology extended to formal ontology views management**. This extension lies, firstly, in the conception of a **view extraction process producing true sub-ontologies** from an ontology. It, secondly, stands in a **change propagation process handling changes impact between the views and the ontology** from which they are derived. The side contributions are: (1) design of an ontological model dedicated to the modelling of changes for $\mathcal{SHOIN}(\mathcal{D})$ ontologies, (2) design of a preventive approach to maintain the structural consistency based on syntactic constraints of $\mathcal{SHOIN}(\mathcal{D})$ before they are applied on the ontology, and (3) the design of a preventive approach to help human resolve logical consistency before changes are applied on the ontology, based on the identification of potential inconsistent axioms sets that each new application of changes could cause.

1.7 Limitations

The main limitations of my proposal lie in its **formalization,** its **functional coverage** and its **implementation**. The methodology is only applicable on ontologies represented in the $\mathcal{SHOIN}(\mathcal{D})$ description logic or any formal language with an equivalent expressiveness (such as OWL). It can, nevertheless, be easily extended to any other formal language by adapting the list of changes to this language list of constructors. From a functional point of view, the method does not handle the change detection phase, which allows experts to understand the need for change of the ontology as a whole. This phase is nevertheless included in the methodology and can be supplemented for example with a semi-automatic ontology learning approach. Also, when propagating changes from one view to the global ontology and from the global ontology to the other views, changes are automatically detected as they are propagated through a change log. The methodology does not propose a method of corrective resolution of

logical consistency dedicated to formal ontologies. When changing or after the change propagation, a posteriori verification of the application of changes on the global ontology is carried out using the Pellet reasoner. However, it is, in principle, only to reinforce the logical consistency preventive resolution process. At the implementation level, the prototype is a Java API extending the Jena ontology management API. It only supports OWL ontologies including OWL DL. It does not propose a GUI to visualize the evolution of an ontology nor the view extraction or the propagation of changes. The ontology and its views are loaded in memory during the change management process, which can cause a scalability problem if the ontology is too complex. Finally, the extraction of the views corresponding sub-ontologies is carried out thought views by SPARQL queries. As this language is dedicated to querying RDF graphs, it causes different problems to implement the extraction of OWL DL axioms and not only RDF triples. A query language dedicated to formal ontologies would be more appropriate.

2 Document Organization

This section gives a brief overview of each of the nine chapters composing this thesis.

2.1 Chapter 2: General Research Field – Formal Ontology

This chapter presents the **general research field** studied in my thesis: **Formal Ontology**. A first section introduces the background related to ontologies, such as: a brief history on their introduction in computer science, their different uses and definitions including formal ontologies. A second section focuses on formal ontologies, their characteristics, their conception and their representation in the context of semantic heterogeneity resolution. According from formal ontology specificities, a third section concludes and deduces three **objectives the proposal** of my thesis should reach: **enable change management of formal ontologies since their creation, enable adaptation of formal ontologies to new user assumptions and the coherent change propagation between a formal ontology and its different assumptions.** From these objectives are developed **eight criteria**. These criteria will serve as guidelines for the two states of art on the Change Management and Ontology Views specific research fields, presented in Chapter 3.

2.2 Chapter 3: Specific Research Fields – OCM & Ontology Views

This chapter focuses on the **two specific research fields** related to my thesis objectives: **OCM** and **Ontology Views**. A first section provides a state of the art on OCM guided by the eight criteria defined in Chapter 2. Its purpose is to expose the founding methodologies of the field but also their limitations, and identify solutions, which are relevant to my thesis objectives. Still considering the eight criteria, a second section studies the ontology views research field by covering a wide range of solutions for the adaptation of ontologies to specific user contexts. A third section concludes and introduces the **three blocks composing the proposal** of my thesis: **(1) designing a collaborative OCM methodology dedicated to $\mathcal{SHOIN}(\mathcal{D})$ formal ontologies, extending it with (2) ontology view specification and $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology extraction**, and **(3) defining the corresponding change impact management process.**

2.3 Chapter 4: $\mathcal{SHOIN}(\mathcal{D})$ Collaborative OCM: Methodology and Model

This chapter introduces and formalizes the methodology and model designed for the **block (1)** of my proposal, i.e. the definition of a **collaborative OCM methodology ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology**. A first section describes the **global OCM methodology, called OntoVersionGraph,** through the different phases of its evolution process. A second section formalizes the **$\mathcal{SHOIN}(\mathcal{D})$ Ontology Model** suited for change management on which my whole proposal is based. A third section formalizes the **change modelling** task of the change management process according to the proposed model. Basic and composed changes are defined as operations adding and deleting axioms from the ontology structure according to the model. The definitions of ontology log of changes and global log of changes, in which changes are represented and traced to ensure OCM features, are formalized.

2.4 Chapter 5: $\mathcal{SHOIN}(\mathcal{D})$ Structural and Logical Consistency Preventive Resolution

This chapter presents the **structural and logical preventive resolution approaches** inspired and based on the $\mathcal{SHOIN}(\mathcal{D})$ model presented in Chapter 4, and respectively used in the change modelling and

change semantics phases of the OntoVersionGraph methodology evolution process. A first section introduces a $\mathcal{SHOIN}(\mathcal{D})$ ontology consistency maintenance background and basic definitions of its structural and logical subtypes. A second section formalizes the structural consistency preventive resolution approach used in the change management methodology during the modelling phase. It is based on **change structural dependencies** and automatically resolves structurally inconsistent changes represented by a user. A third section formalizes the logical consistency preventive resolution approach used to help the user identifying and debugging **minimal logically inconsistent sets of changes** modelled, before implementing them on the ontology. It is based on **axiom unsatisfiability constraints** derived from $\mathcal{SHOIN}(\mathcal{D})$ changes logical constraints.

2.5 Chapter 6: $\mathcal{SHOIN}(\mathcal{D})$ Ontology Views Specification & Sub-Ontology Extraction

This chapter formalizes the block (2) of my thesis proposal, i.e. **modelling the specification of views on $\mathcal{SHOIN}(\mathcal{D})$ ontologies and the logical extraction of the corresponding $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. A first section presents and explains the approach for specifying views on an ontology, according to the following criteria: a view must produce a sub-ontology, so it must describe a smaller domain than the ontology one, while remaining evolvable, versionable and consistent. A second section formalizes the process of **sub-ontology extraction** from a $\mathcal{SHOIN}(\mathcal{D})$ ontology. It also shows how the global log of change of an extracted sub-ontology is built from the ontology global log of changes to support the OCM features.

2.6 Chapter 7: Change Impact Management of $\mathcal{SHOIN}(\mathcal{D})$ Ontology & Views

This chapter formalizes the block (3) of my thesis proposal, i.e. modelling the **impact management process of a consistent change propagation phase** within the OntoVersionGraph evolution process. The impact management process ensures two types of consistent change propagation: first, the propagation from a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views, called the **Top-Down Propagation approach**, and, second, from a sub-ontology defined by a view on a source ontology to this ontology and to the other related views, called the **Bottom-Up Propagation approach**. The Top-Down Propagation approach and the Bottom-Up Propagation approach are respectively formalized in the first and the second sections. A third section illustrates the use of the two approach types by applying the whole methodology on two examples.

2.7 Chapter 8: Prototype Implementation

This chapter describes a **first prototype implementation** of the OntoVersionGraph methodology. A first section gives a general overview on **OntoVersioning**, the Java API prototype dedicated to OWL DL OCM. A second section details the **change management module** implementing the block (1) of the methodology in the API. A third section describes the **view specification and sub-ontology extraction module** implementing the block (2) in the API. A fourth section presents the **change impact management module** implementing the block (3) in the API. These three last sections also bring a discussion about the benchmarks and tests realized for each module to identify the limits of the prototype and the improvements required to go beyond.

2.8 Chapter 9: Conclusions and Future Works

This chapter constitutes the conclusion of the document. A first section summarizes the whole points dealt in my thesis. A second section concludes on a discussion on the contributions and limitations of my thesis proposal and gives a non-exhaustive list of **research perspectives**.

References

Berners Lee, T. (2000). Interview by Ethirajan Anbarasan published in UNESCO's Courier, September 2000.

Berners-Lee, T. (2000a). Semantic Web Stack.

Cocchiarella, N. B. (1991). Formal ontology.

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. International journal of human-computer studies, 43(5), 625-640.

Guarino, N., & Musen, M. A. (2005). Applied ontology: Focusing on content. Applied Ontology, 1(1), 1-5.

Sowa, J. F. (1999). Knowledge representation: logical, philosophical, and computational foundations.

Chapter 2

General Research Field: Formal Ontology

Summary

This chapter presents the **general research field** studied in my thesis: **Formal Ontology**. A first section introduces the background related to ontologies, such as: a brief history on their introduction in computer science, their different uses and definitions including formal ontologies. A second section focuses on formal ontologies, their characteristics, their conception and their representation in the context of semantic heterogeneity resolution. According from formal ontology specificities, a third section concludes and deduces three **objectives the proposal** of my thesis should reach: **enable change management of formal ontologies since their creation, enable adaptation of formal ontologies to new user assumptions and the consistent change propagation between a formal ontology and its different assumptions.** From these objectives are developed **eight criteria**. These criteria will serve as guidelines for the two states of art on the Change Management and Ontology Views specific research fields, presented in Chapter 3.

Plan

1 Ontologies and Formal Ontology 35
1.1 Ontology in Computer Science 35
1.2 Ontology Definitions 36
1.3 Formal Ontology in Applied Ontology Research Field 39
1.3.1 Ontology: Study of Content 39
1.3.2 Formal Ontology Building Process 41
1.4 Conclusion 42
2 Formal Ontology Characteristics 43
2.1 Formal Ontology Composition 43
2.2 Formal Ontology and Interpretation 43
2.3 Formal Ontology and Inference 44
2.3.1 Terminological Level Inference 44
2.3.2 Assertional Level Inference 46
2.4 Formal Ontology Quality 47
2.5 Formal Ontology and Language 48
2.5.1 A Formal Language 48
2.5.2 Formal Language Expressivity 48
2.5.3 Semantic Web Representation Languages 49
2.5.4 Description Logics vs. OWL for Formal Ontology Representation 50
2.6 Discussion 52
3 Conclusion 53

Previous chapter has introduced: the Semantic Web expectations for sharing knowledge bridging semantic heterogeneity of web data, the knowledge evolution problem within a domain where the community shares different assumptions, the general approach envisaged in my thesis to resolve it and its contributions. This chapter focuses on the general research field studied in my thesis, which addresses the creation of shared representation for top-level knowledge or specific domains *:* Formal Ontology. The first section introduces the notion of ontology and focuses on the definition of formal ontologies in computer science. The second section defines the concepts necessary to understand the characteristics of formal ontologies: **composition, interpretation, inference, quality** and **representation language**.

1 Ontologies and Formal Ontology

This section presents the research field by a general discussion. The first part explicates the notion of ontology through a brief history. It describes the expected objectives of its introduction into the computer science field and its common uses. It explains the advantage of its use for the resolution of semantic heterogeneity. The second part of this section discusses the different ontology definitions in computer science. The third part presents the research field dedicated to formal ontologies, Applied Ontology, and describes the design process of such ontologies.

1.1 Ontology in Computer Science

This first part introduces a brief history of the notion of ontology, in order to locate and understand its introduction in computer science. It then describes the various reasons and objectives linked to their common uses.

Since Aristotle, human has wondered how to represent the world around him by studying its properties and categories. This science, called **Ontology**, is a sub-domain of metaphysics. This study was one of the sources of inspiration for artificial intelligence in the early 80's, which borrowed the notion of "**ontology**" to turn it in a representation support for human knowledge. In the 70's, without naming it, the notion already existed in the different systems of knowledge representation. At the time, it was namely declined through the **terminological level of a knowledge base** in description logics. This level, called **T-Box**, describes the terms existing in the **representation of a domain** and their **logical features** in **conceptual graphs** (Sowa, 1999). Conceptual graphs describe the multi-inheritance hierarchies among concepts or relationships and frames schemas (Minsky, 1974). The beginnings of this branch of artificial intelligence merged with computer science ones. Formal representations of artificial intelligence for knowledge have the attraction of being both **understandable by humans** (Quillian, 1968) and **easy to handle by computers**, using rules defined on the symbols of these representations. The term "ontology" only appeared in computer science in the early 80's (McCarthy, 1980) and has been adopted by the entire community. Today, ontologies are at the heart of the study of information systems and the Web. If the Web turns along the lines of the **Semantic Web** as described by Tim Berners-Lee (Berners-Lee, 2001), it will provide services based on ontologies to add a **semantic layer**, a structure on linked Web resources (see Chapter 1 section 1).

The main reason, for the introduction of the term "ontology" in computer science, is the passage of the need to manage data, syntax and structure in computer systems, to the **need to manage knowledge** gained from these data, i.e. their **meaning**. In the early 80's, several observations have shown that the information acquired and accumulated by an organization is not stored or properly transmitted, while it is the main asset of modern organizations (transmission of business skills, know-how, etc.). Much of this knowledge is only held by employees with all the risks that entails. Therefore, the need to **represent, store and share knowledge** was quickly identified. The development of the Internet has

supported this need with the ability to publish this knowledge online, while raising a new problem: **semantic heterogeneity** of data in access, exchange and sharing. Semantic heterogeneity can be defined by the existence of variations in the way information is specified and structured, according to its heterogeneous sources, which give it a **different meaning**. For example, the word "bow" in natural language does not have the same meaning from the point of view of an archer and from the perspective of a violinist. The knowledge represented by the term in both contexts is not the same. There is a need to **disambiguate** potentially heterogeneous information from a semantic point of view. For example, in medicine, doctors have to use the same terms to describe diseases, symptoms etc. Through their **controlled structured vocabularies**, ontologies can explicitly specify the knowledge of a domain of discourse. The identified community of this domain can therefore share the same **terminology** to exchange data. They are able to **better communicate and understand each other**. Moreover, as a computer can easily manipulate ontology, it will be able to reuse its terminology by comparing it to information received, in order to extract its sense and operate it. As seen in Chapter 1, ontologies have been introduced in the Semantic Web research field in the early 2000's to exploit textual documents available on the Web in **formalized information**. As such, they are sometimes presented as tools for knowledge representation adapted to the Web environment, to automatically **transform data into information and information into knowledge.** Practically, an ontology is used to **unify the meaning** of the terms used within a domain, in a consensus obtained by the collaboration of domain experts (Guarino, 1997). This task is called **semantic integration** (Noy, 2004). Semantic integration allows today ontologies to perform the following tasks: acquisition and representation of knowledge; search and retrieval of knowledge: returning knowledge relative to the user's request or discover new knowledge inferred from logical constraints or rules (inference); knowledge sharing and integration: integrating different sources of information; knowledge management; simplification of the man-machine dialogue.

Two basic scenarios exist for the application of semantic integration. The first is the exchange of data between systems of the **same domain** with the **same terminology** but a **different conceptualization**. The second is the exchange of data between systems of the **same domain** with the **same terminology** and the **same conceptualization**. According to (Guarino, 1995), ontologies should be modelled to solve semantic heterogeneity problem of the first scenario. Experience shows that, in practice, they have hitherto been mainly used to share data schemas for the second scenario. This resulted in the creation of "**data silos**" where heterogeneous data, rather than being unified, are stored independently of each other in silos, making them inaccessible between them.

1.2 Ontology Definitions

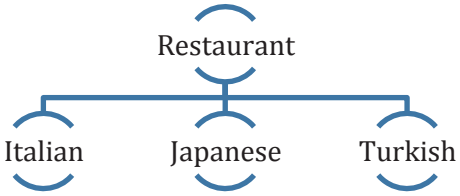
This part discusses the different ontology definitions in computer science.

Several ontology formalisms exist in the literature. They differ according to the domain community identified needs and use. Literature distinguishes, among others, the following definitions:

1. A **conceptual definition of ontology**: "agreement on a shared, and possibly partial, conceptualization" (Gruber, 1993, Smith & Welty 2001).

The scientific community generally agrees on these terms as the definition of a basic ontology. "Agreement" means the consensus among participants on the semantic definition of the heterogeneous elements of the modelled domain. "Shared **conceptualization**" means that the concepts representing knowledge are understandable to all users of the ontology. "Possibly partial" stands for the possibility to **model the whole or only a part of the domain** in the ontology. Indeed, the domain knowledge can be partially modelled to ensure a consensus on the assumptions shared by its users, while putting aside those on which they disagree. A conceptual ontology, **independent of any language**, can be a synonymous of a

semantic conceptual model. The main consequence of such a definition is that a conceptual ontology can accept **multiple representations** according to the language used (Guarino 1995). For example, the following representations "a pizzeria, a sushi bar and a kebab are restaurants" and "an Italian restaurant, a Japanese restaurant and a Turkish restaurant are restaurants", can be represented by a unique conceptualization as follows:



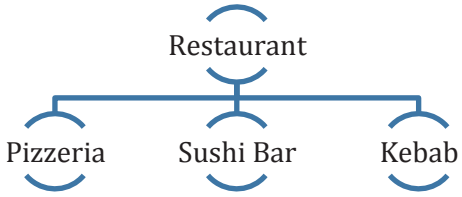
Conceptualization 1: concepts Italian, Japanese and Turkish are all sub-concepts of the concept Restaurant

Figure 2 – One Conceptualisation for Multiple Representations

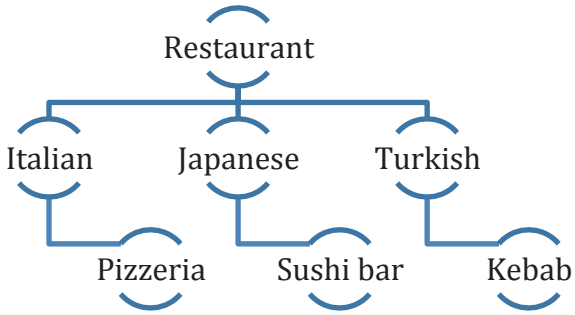
Conceptualization 1 on Figure 2 is intentionally brief to maintain a consensus on what the different representations have in common. The conceptual definition of ontology is suited for **resolving syntactic heterogeneity**. Indeed, if information is specified in syntactically different languages, they can interoperate through a conceptual model independent of any language.

2. A **representation of a conceptualization** definition: "catalogue of the types of things assumed to exist in a domain, from the point of view of a person using a language to talk about the domain" (Sowa, 2000).

According to this definition, the ontology is considered the result of the representation of a conceptualization. It is thus **dependent on a representation language**. This implies that **multiple conceptualizations can have the same representation.** It can be the case if the accuracy is not important, so the ontology does not reflect all the nuances modelled in conceptualization. It can also be the case if the representation language is not expressive enough to distinguish these nuances.



Conceptualization 1: concepts Italian, Japanese and Turkish are all sub-concepts of the concept Restaurant



Conceptualization 2: concepts Pizzeria, Sushi Bar and Kebab are respectively sub-concepts of the concepts Italian, Japanese and Turkish, which are all sub-concepts of the concept Restaurant

Figure 3 – Different Conceptualizations of the same Representation

For instance, the two conceptualizations illustrated by Figure 3 above, although different, can have the same representation in natural language: "A pizzeria, a sushi bar and a kebab are restaurants." This representation is correct because it respects the hierarchy constraints modelled by both conceptualizations. However, it is not sufficiently precise to differentiate them. The definition of an ontology as the representation of a conceptualization, although dependent on a language, does not refer to the use of an explicit logical language. Such an ontology can be used to describe **taxonomies** representing knowledge of a domain in the form of classes of items. It can also model **thesauri**, which organize and control this knowledge using semantic relations and equivalence relations. Taxonomies and thesauri are often specified in **natural language** or other **non-formal languages**. They are used for **structural heterogeneity resolution**. Indeed, their precise knowledge structure allows structurally indexing heterogeneous data.

3. A **formal ontology** definition: "a systematic, formal and axiomatic development of logic for all forms and modes of existence."(Cocchiarella, 1991, Guarino, 1995)

" Systematic development of logic " means that the ontology is based on a **logical system of knowledge**. Ontology is "**formal**", if this knowledge is explicitly represented in a logical language, which does not tolerate ambiguity. For example, it is possible to note that the types of concepts are organized in a partial order by the relation type/sub-type (Sowa, 1999), as opposed to an informal ontology consisting of a catalogue of types that are either undefined, or defined by assertions in natural language. Knowledge is represented as **axioms**, that is to say, obvious and not contradictory propositions respecting the logical constraints, hence the term " axiomatic ". " In all its forms and modes of existence " means that knowledge must be shared and understood by each member of the community, even in **different contexts**. This formal definition of ontology adds an explicit and systematic representation of properties of the ontology model and its elements, to the concept of a shared conceptualization. This representation, describing a clear, precise, and logical knowledge of a domain, allows the community to interoperate, despite the heterogeneity of their assumptions about it. That is why it is suitable to **resolve semantic heterogeneity**. Ontologies represented by description logics and other formal languages often fall into this category.

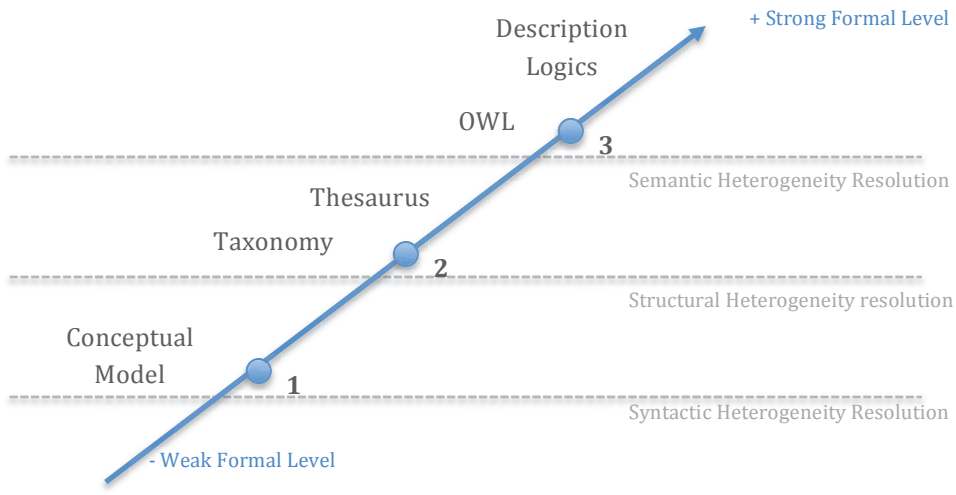


Figure 4 – Ontology Definitions Classifications according to their type of Heterogeneity Resolution

Figure 4 above positions the three ontology definitions (1-3 in Fig. 4) depending on their type of heterogeneity resolution and the accuracy level of specification. It shows that more the specification is restrictive; more the semantics of their domain described are strong. For instance, an ontology of

representation, such as a thesaurus, describes stronger semantics than a conceptual ontology. Indeed, the use of language to represent a conceptualization allows to model more strictly structured knowledge with the language-specific symbols. Semantic relationships between concepts are no longer limited to solely conceptual relations but can be constructed from the symbols and language terms. For example, the relationships "a book is bought by a person" and "a book belongs to a person", defined in natural language, could not be differentiated within a conceptual model. A conceptual model would by default, model a single conceptual relationship with a functional integrity constraint. A formal ontology has stronger semantics than an ontology of representation. This is because a **formal language** is used to explicitly represent and restrict knowledge according to its logical constructs. For example, the vocabulary of the AL[6] description logic, composed of universal and impossible constants, concepts, binary relations (roles), negation, intersection and specialization constructs, universal and existential quantifiers, can formalize the following descriptions: universal concept (T), the impossible concept (T), atomic concepts, negations of atomic concepts, intersection of concepts, value restrictions for concepts, universal and existential quantification restrictions. Axioms such as disjunctions and equivalences may be represented as well as the instantiation of concepts and roles assertions.

The problem addressed in this thesis concerns the maintenance of the semantic interoperability of information systems that share the same domain. The conceptualization of this domain may differ for each system because of the different assumptions they need to share on it. To clearly define the different conceptualizations and unify them within a unique representation, a formal ontology is required. Formal ontologies are studied in the research field called "Applied Ontology" which is presented in the next part.

1.3 Formal Ontology in Applied Ontology Research Field

" Applied Ontology " is a new direction in the field of computer science ontologies recommended by (Guarino & Musen, 2005). It is an **interdisciplinary science** based on **philosophy, cognitive science, linguistics** and **logic**. Its purpose is to **understand, clarify, communicate and make explicit the assumptions of people about the nature and structure of the world**, in order to help them understand each other. This goal is achieved by resolving the semantic heterogeneity that may exist between different conceptualizations of the world. To capture the interest of this approach for this thesis, we will first see why, contrary to data models emphasizing structure, formal ontologies should **focus on the content**. We will then describe the two essential activities identified by the Applied Ontology in the development process of formal ontology.

1.3.1 Ontology: Study of Content

According to Guarino, formal ontology has an inevitable interdisciplinary nature. It is based on the **logical, epistemological, conceptual and linguistical level studies,** which are respectively those of logic, philosophy, cognitive science and linguistics. (Guarino 1994) describes the following table to locate the ontological level compared to these other levels. They are classified according to their level of abstraction from data content.

[6] *AL: Attributive Language. This logic can be considered as the basic logic of the other DLs.*

Abstraction Level	Study Level	Primitives	Interpretation	Main Characteristic
+ ↑	*Logical*	*Predicates*	*Arbitrary*	*Formalization*
	Epistemological	*Structural relations*	*Arbitrary*	*Structure*
	Ontological	***Ontological relations***	***Constrained***	***Signification***
	Conceptual	*Conceptual relations*	*Subjective*	*Conceptualization*
-	*Linguistic*	*Linguistic terms*	*Subjective*	*Language dependency*

Table 1 – Primitives, Interpretation and Main Characteristics of the Ontological Level according to the Other Study Levels.

Logic is used to describe a **formalization**, i.e. the **formal structure of a content** (see Table 1 row 1). It uses predicates, which describe the content properties. For example, the following predicate "being tall", can be used in a proposition noted *beingTall(Peter)*, which describes a property of a content. Predicates allow using logical operators to build logical formulas. But a **logical formula has an arbitrary interpretation**. For example the sentence " Peter is tall or is not tall" does nothing to describe the previous content. It is not conditioned by the content but considers the invariances of the latter, i.e. properties that are true in all reachable states of this content from its original state. The reachability relation describes all the states a content can reach over time or considering different contexts, it does not depend on this content but on the concept referring to this content. Indeed, logic, although very useful for describing the formal structure of a content is independent thereof.

The epistemological level describes **structural relationships between contents** to structure the knowledge of the world in general (see Table 1 row 2). For example, the mereology relationship defines an element of the world as a part of another element. "The heart is a part of the human body" is an example of mereology. Similarly, the subsumption relation is used to define an element as a sub-type of another element. "A man is a sub-type of a human" is an example of subsumption. The epistemological level can provide structural characterization content but is in no way conditioned by it. Structuring relations are relationships independent from content.

The conceptual level, meanwhile, uses **conceptual relationships**, that is to say relations between **concepts** in order to build a **conceptualization of a domain** of discourse (see Table 1 row 4). A concept can be defined as the mental and abstract of a content. A conceptual relation is used to describe a semantic link between two concepts. For example, the concept *HumanBody* can be connected to the concept *Heart* by relationship *haveOrgan*. The conceptualization of a domain corresponds to the set of conceptual relationships that describes it. It represents a general and abstract knowledge of the domain.

The linguistic level is used to **select objects in a domain** of discourse, that is to say, its content, called **references**, referring to **concepts** and using **symbols** of a language, called terms (see Table 1 row 5). It is used to represent a conceptualization in a language. There is a causal relationship between a term, also called **sign**, and a concept, and between this concept and the reference. The reference is the person or thing to which a linguistic expression refers. The sign is the term used to designate this reference in a certain language. The **concept is the meaning of the term**, also called its **intension**. The reference triangle or triangle of meaning (Ogden & Richards, 1923) illustrates this definition. Figure 5 below illustrates it on the left with one of its possible representations on the right.

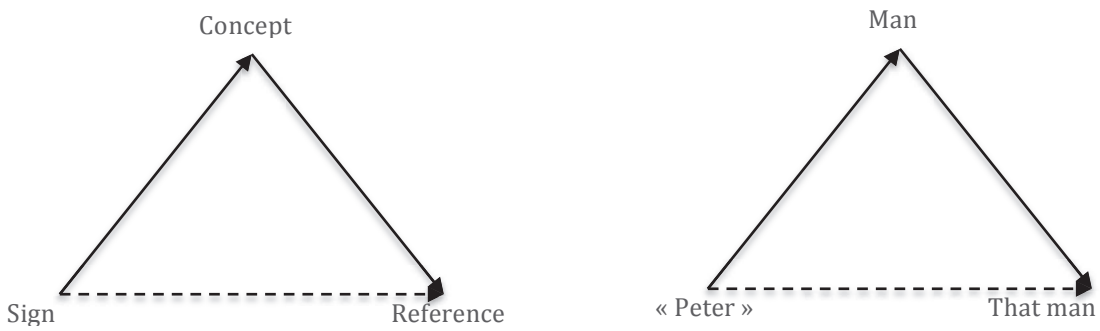


Figure 5 – Reference Triangle (on the left) and a Possible Representation (on the right)

On the reference triangle representation on the right, the concept Man symbolizes the term "Peter." The reference "That human" refers to the concept Man to interpret the term "Peter." Indeed, it is only through the concept Man associated with the term "Peter", that it is possible to interpret this term correctly whatever the situation. The relationship between the term and the designated reference is implicit: it is called interpretation.

The ontological level is at the heart of these different levels. It can **represent, using a structured language, a structured and logically constrained conceptualization of a content**. This representation is called a **terminology**. For example, the following terminology "if Peter is a Man then it is a Human" builds knowledge from a content by describing a logical relationship between the term "Peter is a man" and the concept "Human". This relationship is called an ontological relation (see Table 1 row 3). The interpretation of a terminology is constrained by this ontological relation and corresponds to the meaning of content. The ontological level is then **conditioned by content**.

1.3.2 Formal Ontology Building Process

Figure 6 below, inspired by (Guarino, 1995), describes the building process of a formal ontology according to the Applied Ontology orientation. It allows to view the contribution, from the other study levels described above, for the ontological level.

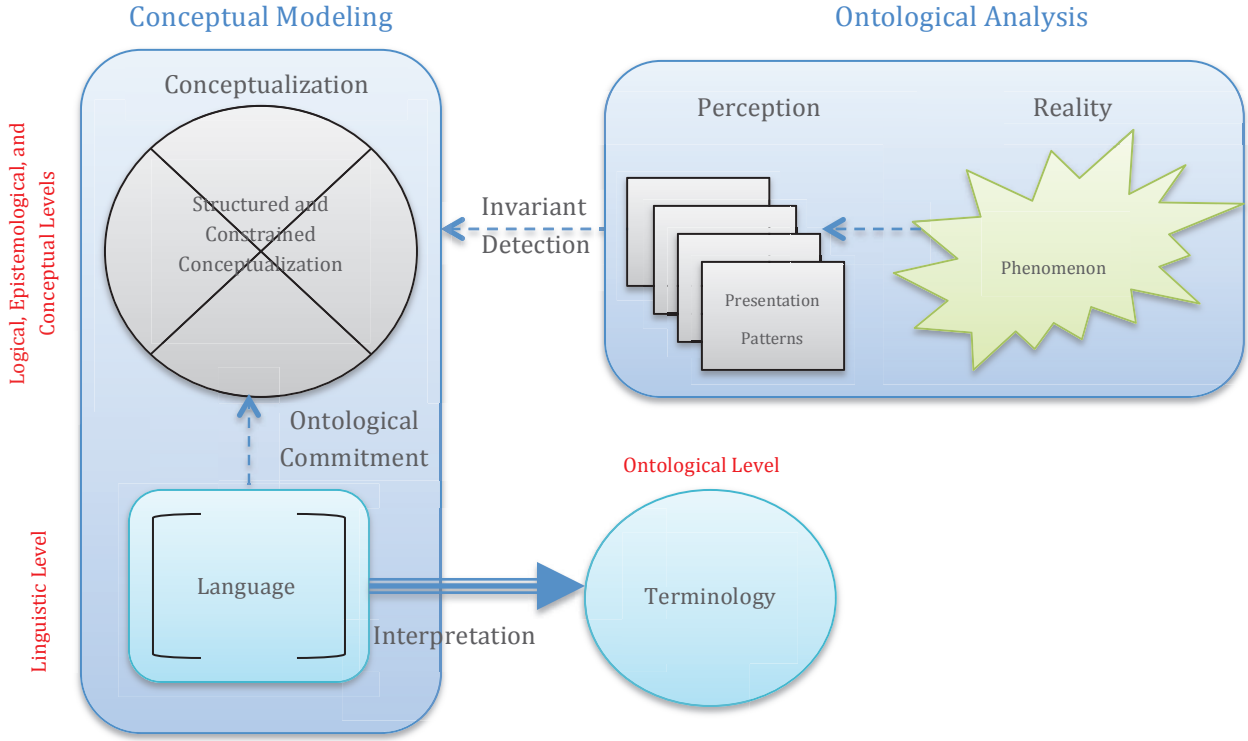


Figure 6 – Formal Ontology Building Process According to Applied Ontology

This process undertakes two activities for formal ontologies development:

1. **Ontological analysis**, which is the **study of the content as such**, that is to say, of the assumptions of the domain, regardless of its representation. It should always be performed before the representation. First, the phenomena of reality in the domain are identified. Then they are perceived and described through presentation patterns. Each pattern describes an assumption supported by a user of the domain on these phenomena. The final set of patterns is the result of the ontological analysis. More numerous they are, more the analysis is complete.
2. **Conceptual modelling**, which is used to **formally describe the results of the ontological analysis**. It compares the different presentation patterns to formally structure the **invariants** therein. The result of this formalization is called a **conceptualization**. A conceptualization is a formal structure of the reality (or a piece of the reality) as perceived and organized by an agent (person, perceptual object), **regardless of the language** used and of the actual occurrence of a specific situation (Guarino & Welty, 1998). It is therefore modelled without representing it in a language. Also all the conceptual relations and structural constraints logically corresponding to the invariants are identified from the presentation patterns comparison (see Fig.6: logical, epistemological and conceptual levels). To produce an ontology, i.e. a terminology (see Fig.6: ontological level), a representation language must then be selected and used to represent this conceptualization (see Fig.6: linguistic level). This language will be used by the domain community to identify referents of this domain. If the conceptualization is verified by these references to the domain content, then the **ontological commitment** is reached.

1.4 Conclusion

This section has addressed the characteristics of the main research field of my thesis. A brief history has shown that the original interest was philosophy since the time of Aristotle. The ontology was then studying the representation of the world by analysing its properties and categories. It has inspired the area of artificial intelligence that was used as a system of representation of human knowledge in the 70's. It appeared in computer science in the 80's, followed by the first fruits of the Semantic Web, to provide a semantic layer to web resources. The primary objective of its use in computer science is the resolution of semantic heterogeneity of information. Semantically annotated data become representable knowledge, understandable by both machines and humans. Different definitions of ontology co-exist and correspond to different objectives and purposes. My thesis subject pointing the problem solving semantic heterogeneity between systems sharing the same field of knowledge, the same terminology but not the same conceptualization, I chose to study the definition of formal ontology. The research field dedicated to formal ontology is Applied Ontology. This field is a new interdisciplinary direction following the philosophy, cognitive science, linguistics and logic study levels. Its approach is based on the study of the content of the domain rather than on its structure, and the various assumptions that people support on it. Its objective is to produce a formal ontology for the domain community members to understand and share its knowledge, and collaborate, while maintaining their own assumptions. This approach is based on two activities: ontological analysis and conceptual modelling. Ontological analysis is the study of assumptions, or points of view, supported users of a domain about its nature and structure. Conceptual modelling allows to formally describing these assumptions for the purposes of understanding and communication.

2 Formal Ontology Characteristics

This second section covers the main characteristics of a formal ontology as defined by Applied Ontology. The first part deals with the composition of a formal ontology. The second explains the notion of interpretation and the third part introduces the notion of inference. The fourth part defines the quality criteria of an ontology and the conditions to qualify it as formal. The last part discusses the ontological representations of languages, including formal languages, and establishes a small state of the art on Semantic Web languages.

2.1 Formal Ontology Composition

This part focuses on the composition of a terminology, issued from the construction process of a formal ontology. It is at least composed the following sets:

- **Non-relational concepts** and **relational concepts** (cf. conceptualization): these are two disjoint sets. The concept *Red*, for example, can symbolize the "red" colour of a reference. This is a non-relational concept. The concept *on*, allows to symbolize a superposed relationship designated by "on" between two objects or persons of a reference. This is a relational concept. Non-relational concepts are often called properties or concepts, while relational concepts are often called relations.
- **Hierarchy of concepts** (cf. structure): relational and non-relational concepts are structured by a hierarchical relationship called **subsumption**. This relationship can be seen as an inheritance relationship by substituting the non-reciprocal relationship " is-a ." E.g. the concept *Man* is subsumed by the concept *Human* because " a man is a human". Conversely, a human is not necessarily a man. Similarly, the relational concept *fatherOf* is subsumed by *isParentOf*.
- **Signatures** (cf. language): they define non-taxonomic semantic relations associated with two functions, the domain and the range, which can be illustrated by *signature(relational concept) = domain × range*. The domain function types the subject of a relational concept with a non-relational concept. The range function types the object of a relational concept with a non-relational concept. For example, in the assertion *"Peter isFatherOf Paul",* the subject of the relational concept *isFatherOf* is "Peter" and the object is *"Paul".* As *"Peter"* and *"Paul"* are both men, one possible signature can be *signature(isFatherOf)=Male × Male*.
- **Axioms** (cf. logic): axioms are obvious and non-contradictory propositions, respecting logical constraints such as domain constraints , conditional constraints and integrity constraints. For example, the relational concept *hasMother* may be constrained by an integrity constraint restricting the number of objects to "one and only one". Thus we cannot form assertions of the form "*Peter hasMother Sylvie* and *Peter hasMother Paulette* ".

2.2 Formal Ontology and Interpretation

This part explains how **interpretation** is done in a formal ontology, namely represented in description logic. As described earlier, the language used to represent the terminology will be also used to interpret it, that is to say, designate referents of the domain, called instances. The set of objects on which the ontology predicates properties and relations defines the **interpretation domain of the ontology**. Among all instances in the interpretation domain, only a portion will be designable by non-relational concepts. The **ability to designate referents of the domain by non-relational concepts** is called **satisfiability**. Therefore, a concept is satisfiable if it can designate at least one instance in the field of interpretation. **Assertions** on instances consist in **interpreting the signatures of the relational-concepts**. Figure 7 below illustrates the concept of interpretation for a formal ontology represented in description logic:

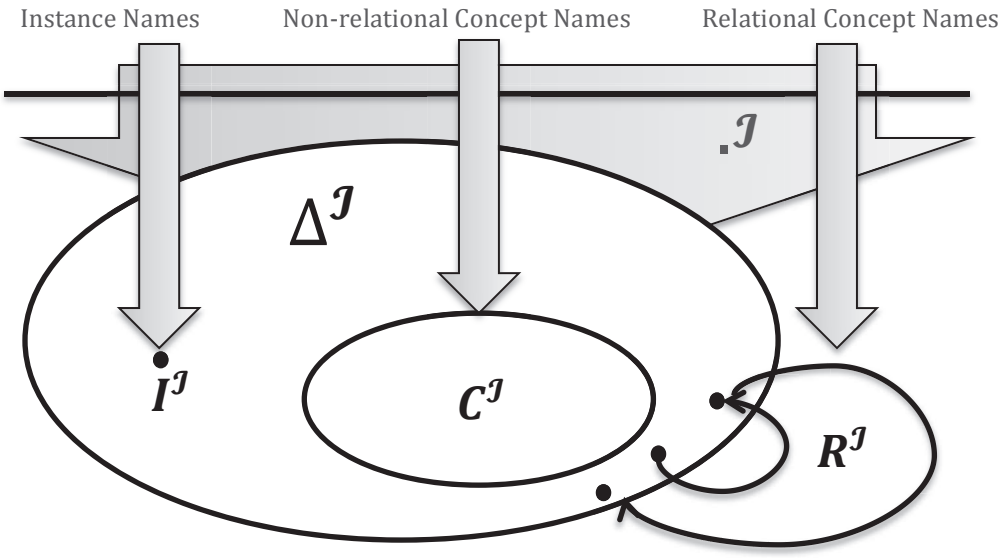


Figure 7 – Description Logic Formal Ontology Interpretation Domain

Interpretation introduces two essential semantic levels for ontologies:

- The **intension**, which is a part of the general meaning corresponding to principle and rules to apply for determining the reference. It corresponds to the terminology.
- The **extension**: which is a part of the meaning corresponding to the effective reference. The extension of the ontology is a subset of the interpretation domain.

An ontology, being a terminology, also corresponds to the intension, unlike the **knowledge base**, which includes both levels. The knowledge base is composed of an assertional component, corresponding to the extension, and a terminology component, corresponding to the intension. The assertional component, also called **A-Box**, reflects the specific (epistemic) states of things. It is designed to solve problems. The terminological component, also called **T-Box**, is independent of the particular states of affairs like an ontology. It is designed to support terminological services. In a knowledge base, the interpretation domain correspond to the set of objects. For instance if I have a knowledge base about a university, it's interpretation domain is the set of professors, students, courses, etc.

2.3 Formal Ontology and Inference

This part briefly introduces the notion of **inference**. Generally inference is a **process of reasoning, which is based on acquired knowledge and revolves around fundamental rules, in order to obtain new information**. This is one of the main objectives of the use of ontologies. Making inferences is deriving new knowledge from axioms described in the terminology. The inference is therefore realized on the terminological level in ontologies. In a knowledge base, using an ontology as a terminology, it is performed both on the terminological and assertional levels, taking into account the individuals.

2.3.1 Terminological Level Inference

Four main inference problems are addressed by the terminological level (Baader, 2003).

- **Satisfiability**: A concept C of a terminology $\mathcal{T}$ is satisfiable, if, and only if, a model $\mathcal{I}$ of $\mathcal{T}$ exists, such that $C^{\mathcal{I}} \neq \emptyset$.

In other words, the interpretation domain $\Delta^{\mathcal{I}}$ admits a non-empty A-Box, such that concept C of terminology $\mathcal{T}$ contains at least one individual for the interpretation model $\mathcal{I}$ (cf. Figure 8).

- **Subsumption**: A concept C is subsumed by a concept D in a terminology $\mathcal{T}$, if, and only if, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for any model $\mathcal{I}$ of $\mathcal{T}$.
- **Equivalence**: *A concept C is equivalent to a concept D for a terminology* $\mathcal{T}$, if, and only if, $C^{\mathcal{I}} = D^{\mathcal{I}}$ *for any model* $\mathcal{I}$ *of* $\mathcal{T}$.
- **Disjunction**: *Concepts C and D are disjoints according to a terminology* $\mathcal{T}$, *if, and only if,* $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$, *for any model* $\mathcal{I}$ *of* $\mathcal{T}$.

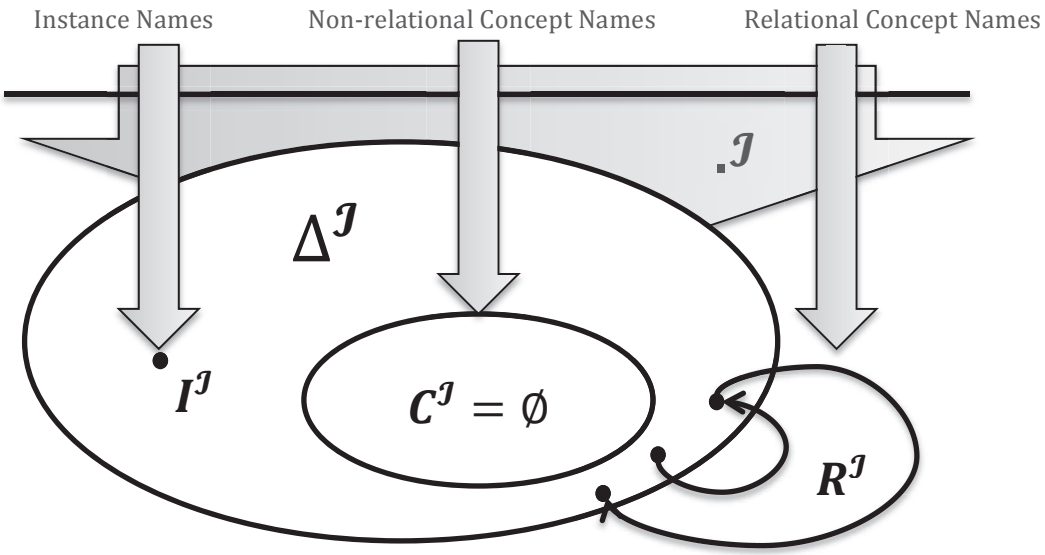


Figure 8. Unsatisfiable Ontology Model

Current inference engines treat these four inference problems as problems of subsumption (see Equation 1 below) or satisfiability problems (see Equation 2 below). Therefore, inference engines dedicated to description logics require one logical inference algorithm for reasoning on the terminological level (Baader, 2003).

$$C \text{ is satisfiable} \iff C \text{ is not subsumed by } \bot$$
$$C \text{ and } D \text{ are equivalent} \iff C \text{ is subsumed by } D, \text{and } D \text{ by } C$$
$$C \text{ and } D \text{ are disjoint} \iff C \sqcap D \text{ is subsumed by } \bot$$

Equation 1. Inference Problem Reduction to Subsumption Problems

$$C \text{ is subsumed by } D \iff C \sqcap \neg D \text{ is insatisfiable}$$
$$C \text{ and } D \text{ are equivalent} \iff C \sqcap \neg D \text{ and } \neg C \sqcap D \text{ are unsatisfiable}$$
$$C \text{ and } D \text{ are disjoint} \iff C \sqcap D \text{ is unsatisfiable}$$

Equation 2. Inference Problem Reduction to Satisfiability Problem

Figure 8 shows an example of unsatisfiable DL ontology model $\mathcal{I}$ containing an unsatisfiable concept C. C is unsatisfiable, because its interpretation is an empty set, which does not admit any individual.

2.3.2 Assertional Level Inference

Assertional level addresses four inference problems (Baader & Nutt, 2003):

- **Consistence**: An A-Box A is consistent w.r.t. a T-Box $\mathcal{T}$ if, and only if, a model $\mathcal{I}$ of A and $\mathcal{T}$ exists.

For example, the set of assertions {*Mother(Mary), Father(Mary)*} is consistent respecting the empty T-Box: since there is no restriction on the interpretation of the concept *Father* and the concept *Mother*, these two concepts can have individuals in common. However, this is no longer true when *Father* and *Mother* are disjoint (see Fig.9).

- **Instance Verification** : Verify by inference if an assertion C(a) is true for every model $\mathcal{I}$ of an A-Box A and a T-Box $\mathcal{T}$.
- **Role Verification** : Verify by inference if an assertion R(a, b) is true for every model $\mathcal{I}$ of an A-Box A and a T-Box $\mathcal{T}$.
- **Recuperation Problem**: Consist, for an A-Box A, a concept C of a terminology $\mathcal{T}$, of inferring the individuals $a_1^{\mathcal{I}} \dots a_n^{\mathcal{I}} \in C^{\mathcal{I}}$ for any model $\mathcal{I}$ of $\mathcal{T}$.

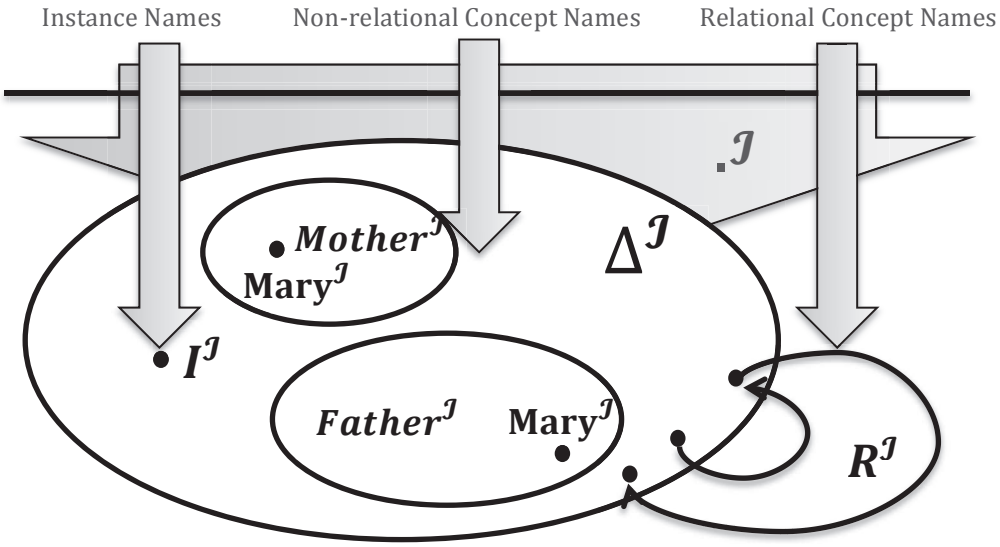


Figure 9. Inconsistent Ontology Model

Inference, both on assertional and terminological levels, is more or less correct depending on the **quality** of the ontology. Indeed, if the terminology is not consistent with presentation patterns identified in the ontological analysis, then its interpretation is not consistent with reality. New knowledge from a wrong terminology will a fortiori be wrong. Similarly, new assertions derived from an interpretation non-conformed to reality will also be wrong. This is why it is important to ensure quality during the development of a formal ontology. According to (Baader, 2003) it is worth emphasizing that, although for convenience, the services for the A-Box are separated, when the T-Box cannot be dealt with by means of the simple substitution mechanism used for acyclic T-Boxes, the reasoning services may have to take into account all of the knowledge base including both the T-Box and the A-Box, and the corresponding reasoning problems become more complex.

2.4 Formal Ontology Quality

This part presents the main criteria for assessing the quality of a formal ontology. The ontology is described by the set of modules of knowledge modelled by the conceptualization and represented by the associated language. Figure 10 below shows that the interpretation domain ontology obtained does not fully correspond to the expected model, for each interpretation. According to the ontological analysis and the chosen language, the interpretation domain of the ontology is more or less in line with the expected one. Ideally, this interpretation model and the expected one should overlap. The assessment of this superposition is used to measure the quality of ontology.

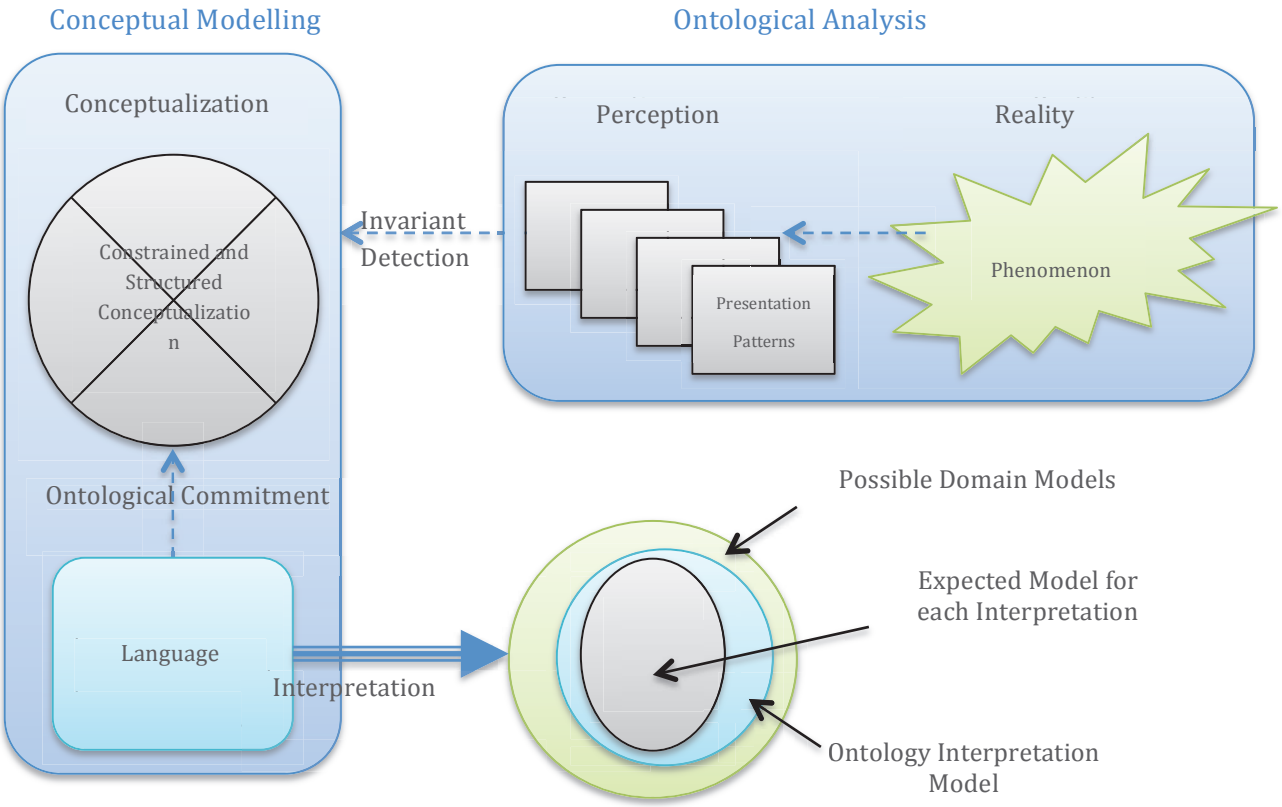


Figure 10– Result of a Formal Ontology Development Process

Quality of a formal ontology is linked to two parameters: precision and correctness of the terminology.

Precision is the distinguishability of the description with respect to the domain. According to (Guarino, 2002) precision is essential when: subtle distinctions are important, recognition of disagreement is important, explanation and thorough justification of the ontological commitment are important, mutual understanding is more important than interoperability. The greater the precision of the ontology is, more the ontology interpretation domain is reduced. Also, conversely, the less the precision is, more this area will be wide.

Correctness refers to the exactness of the description with respect to the domain. It is essential in order to ensure ontological commitment. The more correct the ontology is, the more there are interpretations corresponding to situations of reality. Conversely, the lower the correctness is, the higher the number of interpretations will be unverified by actual situations. Therefore there will be contradictions between the description and reality. Consequently, an ontology with the maximal precision and correctness is described as good ontology. In fact, the whole set of the ontology models overlap the expected one. If, despite maximal precision, correctness is lower, it means that some interpretations

cannot be verified by real-life situations, without being contradicted. The set of its models contains, but exceeds the expected one. The ontology is of less good quality. Also if the ontology has, in contrast, maximal precision but less correctness, this means that some interpretations are inconsistent with reality. The set of its models does not overlap completely and is smaller than the expected one. The ontology is qualified as a bad ontology. Finally, if precision and correctness are minimal, many interpretations cannot be verified and/or are in contradiction with reality. The set of ontology models is too broad and does not contain the expected one in its totality. This leads to a worse ontology.

The level of precision depends on the **exhaustibility and the rigor of the ontological analysis** but can only be guaranteed by the **use of a formal representation language**. The level of correctness depends on the **expressivity of the language**. Formality and expressivity of a language are presented in the next part.

2.5 Formal Ontology and Language

This part helps understanding how a language can be qualified as formal. It discusses a small state of the art on Semantic Web ontology languages and identifies those fitting well with formal ontology representation.

2.5.1 A Formal Language

A **formal language** is a language that uses a set of terms and syntactic rules to allow to unambiguously communicate (as opposed to natural language). An ontology must be explicitly specified to maximize the correctness of the representation with respect presentation patterns. Also, the formality of an ontology language is complementary to the completeness and thoroughness of the list of identified presentation patterns. Indeed, having a formal ontology language is not enough to produce a formal ontology.

2.5.2 Formal Language Expressivity

Ontology precision can only be guaranteed if its representation language is expressive enough. According to (Serafini, 2012), the expressivity of a language can be measured by its **distinguishability**, i.e. by situations that are considered indistinguishable. So to capture the expressivity of a language, we must first find some appropriate **structural invariance between models**. Also the descriptive objective of a language refers to situations in the domain and their structures. The expressiveness of language therefore always refers to a basic set of elements of the domain. This can be either informal elements, which correspond to objects, people, situations, etc., either logic elements or symbols that correspond to a class of mathematical structures in which the language is interpreted. **Symbols are used to represent informal elements**. Thus, there is a binary relationship between the symbols and informal elements. It is called a **satisfiability relation**.

The following example illustrates this relationship with a language using ideograms. The chosen language is an ideographic language called Bliss (Hunnicutt, 1986). It was originally developed by Charles K. Bliss to facilitate the communication of children with physical disabilities. Then it has been widely used, in North American hospitals, to help dumb, paralyzed or paraplegic people to communicate. It is composed of about 4500 symbols, and has its own vocabulary and its own syntax. Learning is progressive and patients at the beginning only use a reduced set of symbols displayed on a tablet. Depending on what they want to express, they choose the symbols and combine them to form sentences. Depending on the number of symbols they have, their communication is more or less accurate. A tablet with 12 symbols as illustrated below (see Fig. 11) allows the patient to express the desire to go home with a combination of three symbols.

Figure 11 – Expressivity of a 12 Symbol Language

However, it is impossible with these three symbols to know whether the patient expresses his own desire or his interlocutor's one, because no symbol to distinguish who he is talking about. So we cannot know if it is a request or a comment. We are also unable to understand if the symbol house is used to talk about his home as a building or his family. The interpretation of these three symbols can thus have multiple meanings in natural language. Indeed, the twelve symbols presented on the tablet are not representative of the diversity of natural language sentences of the given set. It is not possible to differentiate between the terms "I want to go home" and "You like going to the house" with this list of symbols. This corresponds to indistinguishable situations as mentioned above. Also, in order to differentiate them, more symbols are required, we need of a more expressive language, such as the following illustrated below:

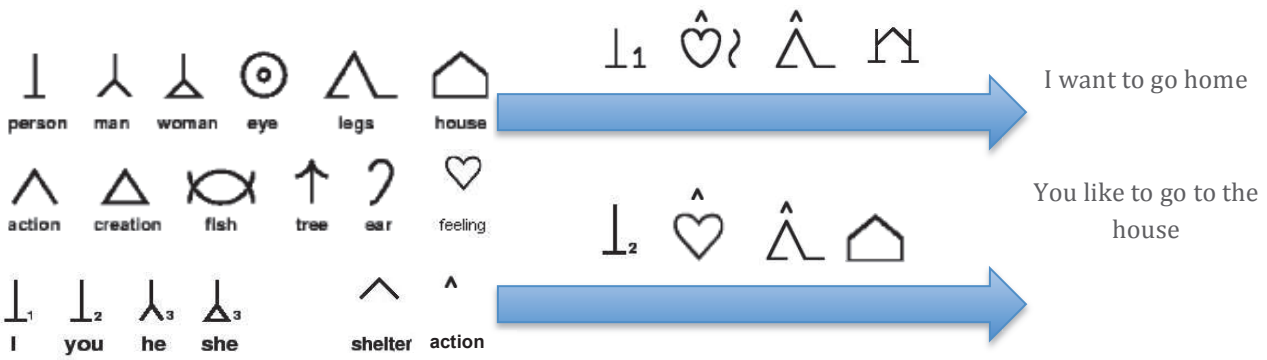


Figure 12 – Expressivity of a 18 Symbol Language

The relationship satisfiability, illustrated in Figure 12 above, is the same as in Figure 15, but it gives better results because the language contains more symbols. Indeed, the **expressivity of a language depends on the number of symbols** it contains, called **constructs**. More expressive the language is, more precise the representation can be.

2.5.3 Semantic Web Representation Languages

The choice of a specific expressivity level, for an ontology representation language, depends on the **granularity level required** for the description of a domain. Many representation languages exist in the literature. Some are specifically used in the Semantic Web. However they are neither all formal nor expressive enough to describe formal ontologies. Following table lists the usual Semantic Web representation languages, in the ascending order of their expressivity. The second column shows the number of constructs, each language contains. The third column gives each type of language and the fourth describes their uses.

Name	Expressivity (number of constructs)	Type	Use
RDF	2	Data Model with no specific syntax	1 Describes metadata for Web resources 2 Used for general descriptions 3 Encode structural information 4 Universal data exchange format
RDF-S	6	Extensible Language for knowledge representation based on RDF	• Provides base elements for the representation of light ontologies or vocabularies used to structure RDF metadata
Description Logics (DLs)	$\mathcal{ALC}$: 16 $\mathcal{S}$: 17 $\mathcal{S}$ Derived DLs: 17 → 31	Family of Logics dedicated for knowledge representation and inference	• Provides semantics and inference services for knowledge bases
OWL	OWL-Lite: 20 OWL-DL: 22 OWL-Full: 22 OWL 2: 29	Knowledge Representation Language based on RDF-S and DLs	• Defines structured and inferable Web ontologies • Data Format for certain DLs

Table 2 – Usual Semantic Web Representation Languages

According to this table, **RDF** (Resource Description Framework) language is very poorly expressive. It is used for Web resources description, but its low expressivity limits the resolution of the heterogeneity to **simple metadata comparison**.

RDF-S (Resource Description Framework Schema) language is a few more expressive. It features the subsumption construct (concept hierarchy, role hierarchy), which allows it to define **vocabularies, taxonomies** and **light ontologies**. However, this language is inadequate for the representation of complex ontologies such as a formal ontology. For example, it is impossible to define a disjunction between two concepts. It is not recommended for the description of ontologies for inference.

Description Logics (DLs) and OWL (Web Ontology Language), respectively containing up to 31 and 29 constructs are highly more expressive. Their logical foundation makes them **suitable for formal ontology** description.

2.5.4 Description Logics vs. OWL for Formal Ontology Representation

DLs are a family of logics dedicated to knowledge description in the form of **labelled directed graphs.** In particular, they allow the inference due to their **logical foundation** and t**heir expressivity is expandable by addition of constructs**. Each DL is a combination of different constructs from a set of 31 constructs. The DL ALC, also called Attributive Concept Language with Complements, contains 16 constructs. It is qualified as the minimal DL, in the sense that a less expressive logic is of poor interest in in knowledge representation for inference purposes. ALC is the basis of all practical DLs extending its set of constructs with others. Each DL has therefore its own a level of expressivity and its own reasoning complexity based on this expressivity. Indeed, **more the DL is expressive, more inference will be complex to achieve.** It is therefore important to assess the level of expressivity that an ontology needs by balancing the inference needs. Table 3 gives a non-exhaustive overview of the reasoning complexity in terminological and assertional levels of a knowledge base, depending on the expressivity of different DLs (Zolin, 2013). This table is constructed from the complexity class definitions (Papadimitriou , 1994) .

- **P**: the class of decision problems taking as input a problem statement and outputting a positive or negative response (yes or no , 0 or 1, true or false). It requires a polynomial time relative to the size of the problem to obtain a solution using a deterministic Turing machine. The problem is qualified as polynomial and has a complexity of $O(n^k)$, for a certain k.
- **NP**: the class of problems that require a polynomial time to find a solution with a non-deterministic Turing machine. Calculations for a deterministic Turing machine form a sequence, while the calculations of a non-deterministic Turing machine form a tree, in which each path corresponds to a sequence of possible calculations.
- **PSpace** : the class of decision problems that require a polynomial amount of memory to solve a problem with a deterministic or a non-deterministic Turing machine.
- **EXPTIME** : the class of decision problems resolvable by a deterministic Turing machine in an exponential time relative to the size of the problem.
- **NEXPTIME** : the class of decision problems resolvable by a non-deterministic Turing machine in an exponential time relative to the size of the problem.

Complexity	**Language**				
	$\mathcal{ALC}$	$\mathcal{S}$	$\mathcal{SH}$ /$\mathcal{SHIF}$	$\mathcal{SHOIN}$	$\mathcal{SROIQ}$
Satisfiability	PSpace	PSpace	ExpTime	NExpTime	NExpTime-Difficult
Consistency	PSpace		ExpTime	ExpTime	NExpTime-Difficult

Table 3. Complexity Classes According to Description Logics Expressivity Level

Note that P ⊆ NP ⊆ PSpace ⊆ ExpTime ⊆ NExpTime (Papadimitriou, 1994). The complexity class of a DL is a determining characteristic for the definition of a knowledge base and by extension an ontology. It qualifies its **decidability level**. Decidability refers to the **existence of an effective method for resolving decision problems specified by logic**. PSpace logics have a maximal decidability, while logics of a complexity class higher than NExpTime may not be decidable.

In order to simplify the process of choosing between expressiveness and complexity, the OWL language, standardized by W3C since 2004, provides three sub-languages corresponding to three levels of expressiveness possible inspired DLs. These three sub-languages, called **OWL Lite , OWL DL** and **OWL Full**, overlap in the sense that : every consistent OWL Lite ontology is a consistent OWL DL ontology, each consistent OWL DL ontology is a consistent OWL Full ontology, each valid OWL Lite conclusion is a valid OWL DL conclusion and each valid OWL DL conclusion is a valid OWL Full conclusion. OWL Lite, corresponding to the DL $\mathcal{SHIF}$, is less expressive than the others but provides full decidability. Unlike OWL Full, corresponding to the DL $\mathcal{SHOIN}(\mathcal{D})$, which has a maximum expressiveness but is not decidable. OWL- DL, is a compromise between these two sub-languages Indeed, OWL DL supports maximum expressiveness while retaining computational completeness and decidability. OWL DL is thus the preferred language and the most widely used for the representation of formal ontologies (Staab & Studer, 2010). Standardization of OWL has prompted the development and/or the adaptation of a number of **reasoners**, including Fact++ (Tzarkov & Horrocks, 2006), Pell[et] (Sirin & al, 2007), Racer (Haarslev, & Müller, 2001), and Hermit (Shearer & al, 2008) and ontology editors such as Protégé (Knublauch, 2004) and Swoop (Kalyanpur & al, 2006). OWL ontologies are developed in areas as diverse as e-science, medicine, biology, geography, astronomy, defence, automotive industries and aerospace. Despite the success surrounding OWL, many contexts where the language has been applied revealed some shortcomings in the original design (Grau & al, 2008). Experience has shown that OWL DL , although the most expressive but still decidable sub-language of the OWL family, lacks several constructs that are often required for complex domain representation. Also difficulties have been reported with respect to its syntax including its alignment with description logics or its translation into RDF. Other limitations in the definition of meta-models, import and versioning of ontologies, annotations, language semantics and

validation of the level of expressivity, lead to extend OWL with new constructs to solve these problems. **OWL 2** corresponds to this extension and was recommended by W3C in December 2012. This language is more expressive than OWL-DL, its expressivity is equivalent to the DL SROIQ, but it is still decidable. Although the reasoning complexity is twice higher - N-ExpTime difficult for OWL 2 against 2N-Exptime for OWL DL - this source of complexity is well understood and should not increase on typical practical problems. Finally, existing reasoners can be and have easily been extended to the DL $\mathcal{SROIQ}(\mathcal{D})$, which thus seems to provide a good logical foundation OWL 2. OWL 2 also includes three sub-languages called profiles: **OWL 2 EL, OWL 2 QL** and **OWL 2 RL**. Unlike OWL, they are not built according to their expressivity level, but suited for very specific uses. Other profiles can be derived further. They are defined by imposing restrictions on the functional syntax of OWL 2. An ontology represented by one of these profiles is a valid OWL 2 ontology. OWL 2 EL is an OWL 2 fragment based on EL++ description logics family. These were designed to enable efficient reasoning with large terminologies. The most interesting is the reasoning service classification, that is to say calculating the relationship subclass among all classes in the ontology. OWL 2 QL was created to allow easier access by queries stored in databases. It is based on the DL-Lite family of description logics. This family has been designed for efficient reasoning with large quantities of data structured according to relatively simple schemas. The main reasoning service profile is the result of conjunctive queries: Given an ontology O and a conjunctive query q, the problem is to calculate all the instances tuples, which are a result of q according to O. OWL 2 RL is a subset of rules of OWL 2. OWL 2 RL was designed so that many reasoning tasks can be implemented as a set of rules in a system of forward-chaining rules. To achieve this criterion of applicability, OWL 2 RL is not as expressive as OWL 2. This profile is interesting in situations where a limited extension of RDF-Schema is desired.

DLs, OWL DL and OWL 2 are the languages of the Semantic Web best able to represent formal ontologies. This thesis, which began before the recommendation of OWL 2, focuses on ontologies specified with the DL $\mathcal{SHOIN}(\mathcal{D})$, equivalent to OWL DL expressivity. It is on this specification that all research work presented in this paper is based.

2.6 Discussion

This section has addressed the characteristics of formal ontologies. The composition of a formal ontology can be summarized in a set of axioms, also called terms, built upon relational and non-relational concepts specified by the symbols of a formal language. Therefore a formal ontology is a terminology. The interpretation of the terminology is used to designate referents of the described domain, called instances. The set of feasible interpretations of the ontology forms its interpretation domain. In a knowledge base where an ontology is used as terminological component, the interpretation domain corresponds to the assertional component consisting of assertions on instances. Inference can be achieved, whether on terminological or on assertional level in a knowledge base. It allows to deduce new knowledge respectively from the axioms of the terminology or from the assertions. Inferences are more or less accurate depending on the quality of the ontology. Quality criteria are the precision and the correctness of the terminology according to the described domain. To satisfy these criteria, a formal ontology, in addition to the rigor and thoroughness of his ontological analysis, requires a sufficiently expressive formal language. However inference is more or less possible depending on the complexity of language, which increases with its expressivity level. Also Semantic Web provides several representation languages with different expressivity levels. Only DLs or languages derived from these logics such as OWL and OWL2, can satisfy these characteristics to represent a formal ontology.

3 Conclusion

This chapter has addressed the general research field of the thesis, Formal Ontology, through the point of view of the Applied Ontology orientation.

The first section has first introduced ontologies and demonstrated their importance in solving the semantic heterogeneity of information systems. It showed that my subject is more specifically related to the problem of heterogeneity of conceptualization addressed by formal ontologies. According to Applied Ontology, resolving this issue, should allow users of a domain to understand, share knowledge and collaborate while maintaining their own assumptions on this domain. We have seen that only a formal ontology can guarantee that resolution.

The second section has addressed the characteristics of a formal ontology: its terminological composition, its interpretation namely in a knowledge base, its inference services, which allow inferring new knowledge from the terminological and assertional levels, its quality assessment according to its precision and correctness and the formal nature of its representation language. DLs or formal languages derived from them, such as OWL and OWL 2, are the Semantic Web best suitable languages for the representation of a formal ontology. This is why this thesis specifically focuses on the DL $\mathcal{SHOIN}(\mathcal{D})$.

According from these specificities, the proposal of my thesis should produce a formal methodology for managing evolution and versioning of a formal ontology and its user assumptions on the domain. It has to reach the three following objectives: (1) enable change management of formal ontologies since their creation, (2) enable adaptation of formal ontologies to new user assumptions and (3) the coherent change propagation between a formal ontology and its different assumptions. In order to ensure the integrity of a formal ontology during its lifecycle, this methodology should meet the following eight criteria:

1. Proposing a formal ontology evolution methodology to enable conceptual modelling subsequent to an ontological analysis,
2. Specifying ontology and changes in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal language,
3. Managing formal ontologies representing all the user assumptions on its domain,
4. Managing the logical consistency, i.e. related to axioms, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages,
5. Management of structural consistency, i.e. related to the representation language, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages with formal representation changes,
6. Evolution and Versioning of a formal ontology and its user assumptions since its creation,
7. Impact analysis & change propagation applied to an assumption to all the other assumptions described by the ontology,
8. Adaptation of the ontology to a new user assumption.

References

Baader, F. (Ed.). (2003). The description logic handbook: theory, implementation, and applications. Cambridge university press.

Baader, F., & Nutt, W. (2003, January). Basic description logics. In Description logic handbook (pp. 43-95).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific American, 284(5), 28-37.

Cocchiarella, Nino B. (1991). Formal ontology. In Hans Burkhardt & Barry Smith (eds.), Handbook of Metaphysics and Ontology. Philosophia Verlag. 640--647.

Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., & Sattler, U. (2008). OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web, 6(4), 309-322.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. Knowledge acquisition, 5(2), 199-220.

Guarino, N. (1994). The ontological level. Philosophy and the Cognitive Science. Hölder-Pichler-Tempsky, Vienna, 443-456.

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. International journal of human-computer studies, 43(5), 625-640.

Guarino, N. (1997). Understanding, building and using ontologies. International Journal of Human-Computer Studies, 46(2), 293-310.

Guarino, N., & Welty, C. (1998). Conceptual modeling and ontological analysis. The AAAI-2000 Tutorial, URL: http://www. cs. vassar. edu/faculty/welty/aaai-2000.

Guarino, N. (2002). Ontology-driven conceptual modelling. In Proc. of the 21st International Conference on Conceptual Modeling, LNCS (Vol. 2503).

Guarino, N., & Musen, M. A. (2005). Applied ontology: Focusing on content. Applied Ontology, 1(1), 1-5.

Haarslev, V., & Müller, R. (2001). RACER system description. In Automated Reasoning (pp. 701-705). Springer Berlin Heidelberg.

Hunnicutt, S. (1986). Bliss Symbol-to-Speech Conversion:" Blisstalk. Journal of the American Voice I/O Society, 3(June).

Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., & Hendler, J. (2006). Swoop: A web ontology editing browser. Web Semantics: Science, Services and Agents on the World Wide Web, 4(2), 144-153.

Knublauch, H. (2004, September). Ontology-driven software development in the context of the semantic web: An example scenario with Protege/OWL. In Proceedings of 1st International Workshop on the Model-Driven Semantic Web.

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. Artificial intelligence, 13(1), 27-39.

Minsky, M. (1974). A Framework for Representing Knowledge. Massachusetts Institute of Technology, A.I. Lab, Cambridge, Massachusetts.

Minsky, M. L., & Minsky, M. (1968). Semantic information processing (Vol. 142). Cambridge, MA: MIT press.

Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. ACM Sigmod Record, 33(4), 65-70.

Ogden, C. K., & Richards, I. A. (1923). The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism. Magdalene College, University of Cambridge.

Papadimitriou C. H. (1994) . Computational Complexity (Addison Wesley, Reading, MA)

Serafini, L. (2012). Expressive Power of Logical Languages, First Interdisciplinary Summer School on Applied Ontology, FBK-IRST, Trento.

Shearer, R., Motik, B., & Horrocks, I. (2008, October). HermiT: A Highly-Efficient OWL Reasoner. In OWLED (Vol. 432).

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web, 5(2), 51-53.

Smith, B., & Welty, C. (2001). Ontology: Towards a New Synthesis.

Sowa., J. F. (1999). Knowledge representation: Logical, philosophical and computational foundations. Pacific Grove, CA USA: Brooks Cole Publishing Co.

Sowa, J. F. (2000). Ontology, metadata, and semiotics. In Conceptual structures: Logical, linguistic, and computational issues (pp. 55-81). Springer Berlin Heidelberg.

Staab, S., & Studer, R. (2010). Handbook on ontologies. Springer.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: System description. In Automated reasoning (pp. 292-297). Springer Berlin Heidelberg.

Zolin , E. (2013). Description Logics Complexity Navigator. URL: http://www.cs.man.ac.uk/~ezolin/dl

Chapter 3

Specific Research Fields: OCM & Ontological Views

Summary

This chapter focuses on the **two specific research fields** related to my thesis objectives: **OCM** and **Ontology Views**. A first section provides a state of the art on OCM guided by the eight criteria defined in Chapter 2. Its purpose is to expose the founding methodologies of the field but also their limitations, and identify solutions, which are relevant to my thesis objectives. Still considering the eight criteria, a second section studies the ontology views research field by covering a wide range of solutions for the adaptation of ontologies to specific user contexts. A third section concludes and introduces the **three blocks composing the proposal** of my thesis: **(1) designing a collaborative OCM methodology dedicated to $\mathcal{SHOIN}(\mathcal{D})$ formal ontologies, extending it with (2) ontology view specification and $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology extraction**, and **(3) defining the corresponding change impact management process.**

Plan

1 Ontology Change Management 60
1.1 Ontological Changes 60
1.1.1 What is an Ontological Change? 60
1.1.2 Ontological Change Types Classification 61
1.2 Ontology Evolution 63
1.2.1 Ontology Evolution Definitions 63
1.2.2 Ontology Consistency Maintenance 64
1.3 Ontology Versioning: 66
1.3.1 Ontology Versioning : Definitions 66
1.3.2 Representing and Tracing Changes for Versioning Purposes 67
1.4 Ontology Change Management 68
1.4.1 Klein and Stojanovic's Methodologies 68
1.4.2 Related Works 69
1.5 Discussion 70
2 Ontology Views 73
2.1 Ontology Views in the Literature 73
2.2 View Specifications Approaches 74
2.2.1 Query Language Approaches 74
2.2.2 Manual, Structural and Logical Approaches 76
2.2.3 Other Approaches: Rules, Pipes and NamedGraphs 76
2.3 Discussion 77
3 Conclusion: Deadlocks and Approach of my Proposal 80

Previous chapter has brought the specificities of formal ontologies, my thesis should undertake. It has concluded my thesis proposal should produce a formal methodology for managing evolution and versioning of a formal ontology and its user assumptions on the domain, reaching the three following objectives: (1) enable change management of formal ontologies since their creation, (2) enable adaptation of formal ontologies to new user assumptions and (3) the coherent change propagation between a formal ontology and its different assumptions. In order to ensure the integrity of a formal ontology during its lifecycle, it has proposed the following eight criteria:

1. Proposing a formal ontology evolution methodology to enable conceptual modelling subsequent to an ontological analysis,
2. Specifying ontology and changes in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal language,
3. Managing formal ontologies representing all the user assumptions on its domain,
4. Managing the logical consistency, i.e. related to axioms, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages,
5. Management of structural consistency, i.e. related to the representation language, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages with formal representation changes,
6. Evolution and Versioning of a formal ontology and its user assumptions since its creation,
7. Impact analysis & change propagation applied to an assumption to all the other assumptions described by the ontology,
8. Adaptation of the ontology to a new user assumption.

Most of these issues are addressed in the following research areas: Ontology Change Management (OCM) and Ontological Views. These criteria are the guidelines of the two states of art presented in this chapter.

1 Ontology Change Management

This first section presents a state of the art on the OCM. The first part introduces and defines ontological changes. The second part presents the activity related to their consistent application, called ontology evolution. The third part describes the ontology versioning activity, managing the different ontology versions issued from ontology evolutions. The fourth part deals with ontology change management (OCM), which is the combination of these two activities from the point of view of (Klein, 2004). It compares several OCM methodologies proposed in the literature from this definition w.r.t. the eight criteria listed above. The last part consists in a discussion reviewing the remaining target issues to be studied for my thesis.

1.1 Ontological Changes

This part presents an analysis on ontological changes. It defines ontological changes and classifies their different types encountered in the literature.

1.1.1 What is an Ontological Change?

An **ontological change** is generally defined as a **modification applied on an ontology**. (Klein, 2004) clarifies this definition by describing an ontological change as an action on an ontology whose result is an ontology different from the original version. This broad definition has been adopted in a recent study on ontological change (Flouris & al 2008). According to the study, performing an ontological change amounts to making modifications on an ontology in response to a need for change, as well as realizing their implementation and managing their effects. In this definition, the need to change the ontology can take

several forms, including but not limited to, discovery of new knowledge, change of focus or point of view on the conceptualization, integration of external sources information, change in the domain, new needs for communication between heterogeneous information sources or ontologies, merging information from different ontologies etc. Three change causes are identified: changes can emanate from a change in the **domain** (the world described by the ontology), in the **conceptualization** (semantically modelling the domain) or in the **specification** (representation language). This definition covers many areas of research, which are often studied separately in the literature but are sometimes strongly related. Among them appear ontology mapping, ontology morphism, ontology alignment, ontology articulation, ontology translation, ontology evolution, ontology versioning, ontology integration and ontology merging. To differentiate but also to identify the relationships between these different areas, the study proposes a classification of types of change.

1.1.2 Ontological Change Types Classification

(Flouris, & al, 2008) proposes a classification for ontological changes, taking into account the various works that have been made in the literature to implement methods, algorithms and applications to manage these changes in existing ontologies. Authors consider that the list of change types of this classification includes all the changes applicable on an ontology. The following list explains the meaning of each label of types of change and proposes a classification of these types according to the three change causes mentioned previously (change in the domain, conceptualization or specification).

- **Ontology Mapping** (Kalfoglou & Schorlemmer 2003): solves problems of interoperability between heterogeneous ontologies. This change takes as input two heterogeneous ontologies and creates a mapping between their vocabularies, by identifying links between their entities.
- **Ontology Morphism** (Kalfoglou & Schorlemmer 2003): solves problems of interoperability between heterogeneous ontologies. This change takes as input two heterogeneous ontologies and creates a mapping between their vocabularies and their axioms, by identifying the links between their entities and axioms.
- **Ontology Matching** (De Bruijn et al. 2004), (Choi et al. 2006), (Euzenat et al. 2004), (Euzenat & Shvaiko, 2007), (Kalfoglou & Schorlemmer 2003), solves problems of interoperability between heterogeneous ontologies. This change takes as input two heterogeneous ontologies and creates a relationship between their vocabularies, by identifying links between entities. This relation is called an ontology alignment.
- **Ontology Articulation** (Kalfoglou & Schorlemmer, 2003): solve problems of interoperability for heterogeneous ontologies. This change takes as input two heterogeneous ontologies and creates an intermediate ontology between them and a mapping between the vocabulary of each of them and the intermediate one. As for Ontology Matching, the intermediate ontology is equivalent to a relationship and identifies the links between the entities of their vocabularies.
- **Ontology Translation(1)** (Dean & al, 2004): translates the ontology into a new representation language. This change takes as input an ontology and a new representation language and creates a new version of the ontology in the new language. This change should produce a semantically equivalent ontology if possible.
- **Ontology Translation(2)** (Kalfoglou & Schorlemmer, 2003) : implements a mapping vocabulary. This change takes as input an ontology mapping and creates a new version of the ontology with new vocabulary mapping.
- **Ontology Evolution** (Stojanovic & al, 2002): addresses the need to change the ontology domain or conceptualization. This change takes as input an ontology and a set of ontological operations and applies these operations on the ontology.

- **Ontology Debugging** (Schlobach & Cornet, 2003) : restores the consistency of an ontology. This change takes as input an inconsistent ontology and makes it consistent.
- **Ontology Versioning** (Klein & Fensel, 2001): provides a transparent access to the different versions of an ontology (including the version you wish to access). This change supports different versions of ontologies as input and creates an ontology versioning management system using the id of its ontology version to identify the versions and determining compatibility between them.
- **Ontology Integration** (Pinto & al, 1999): merge the knowledge of several ontologies describing similar domains. This change takes as input two ontologies and merges them to create an ontology covering a wider area.
- **Ontology Merging** (De Bruijn & al, 2004): merge the knowledge of several ontologies describing the same domain. This change takes as input two ontologies covering the same domain knowledge and merges the two ontologies to create a most accurate and comprehensive ontology of this domain.

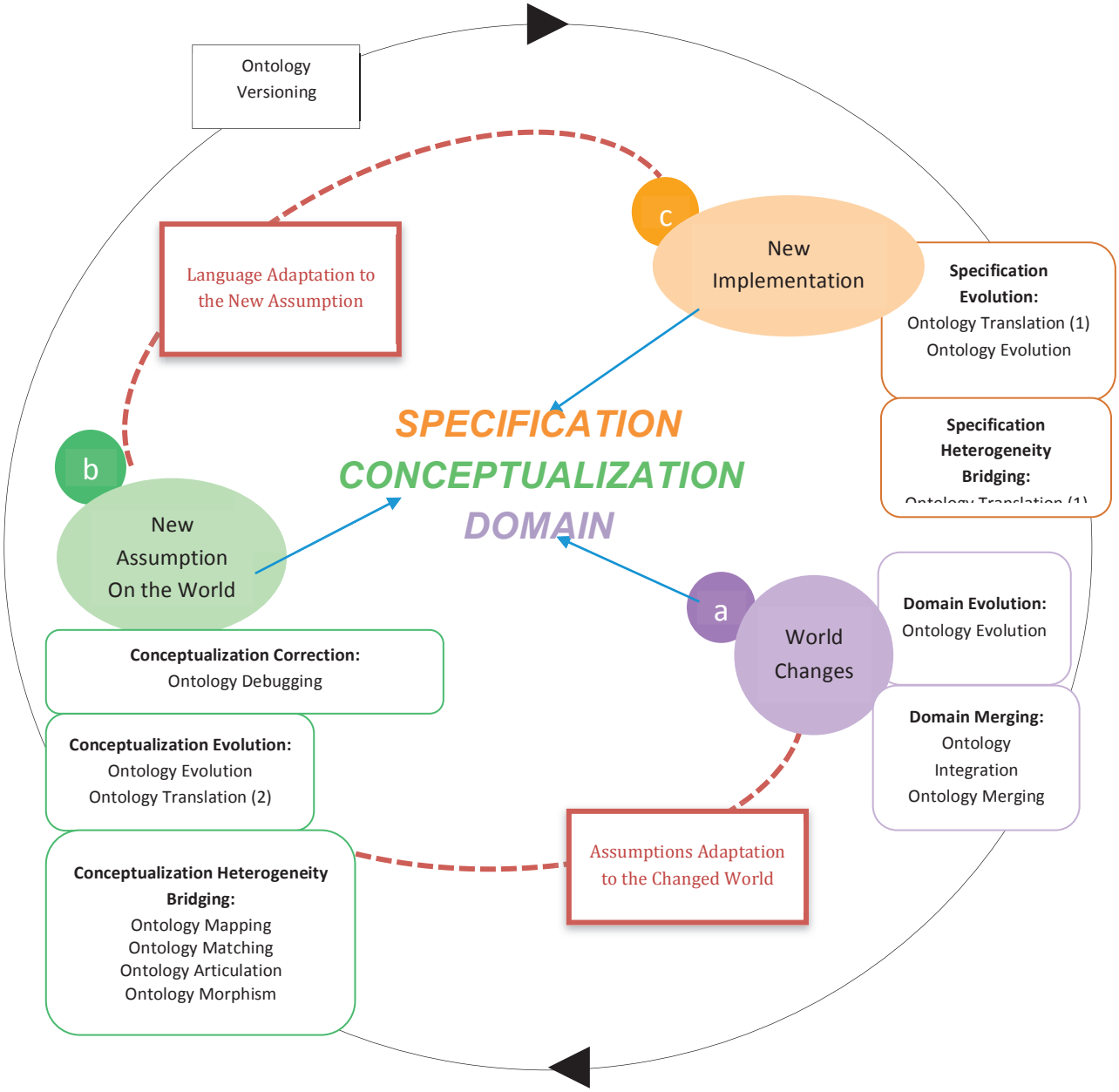


Figure 13: Change Types Classification According to the Three Causes of Changes of an Ontology

Figure 13 above classifies the different types of changes identified by (Flouris & al, 2008) according to the three types of change causes and the definitions of different types of changes. A need of change in the domain can result in a fusion or evolution of the domain. A need of change in the conceptualization can imply an evolution of the conceptualization, bridging the conceptualization heterogeneity or a correction of the conceptualization. A need of change in the specification may involve an evolution of the specification, bridging specification heterogeneity. This allows as well to visualize the **impact** that each type of change may have on the definition of the ontology and to understand the other changes that their application may imply. In particular, a change identified in the domain (a in Fig.13) may require one or more changes in the conceptualization (b on Fig.13), and even, but less often, a change in the conceptualization may involve a change in specification (c on Fig.13) .

Therefore the links that may exist between different types of changes can be described. For example, integration and merging ontologies being changes to meet a need in the domain of ontology have an undeniable impact on the conceptualization of the merged ontology. They bridge heterogeneity of conceptualizations by using mapping techniques, matching, alignment, and/or morphism. Also integration and merging of ontologies specified in different languages may impact the specification of the merged ontology by implying the need to translate ontologies in the same language (Ontology Translation (1) on Fig.13). The description of these links also allows understanding that wherever a change is applied, on the conceptualization or on the specification, **evolution of the ontology** is always required (Conceptualization Evolution and Specification Evolution on Fig.13). It is necessary in the sense that the ontology impacted by these changes will be changed by their application. **Ontology versioning** is also required to produce and manage versions of the ontology generated for each evolution. Both types of changes are therefore involved throughout the life cycle of the ontology and are at the heart of **ontology change management (OCM)**.

Based on these conclusions, the following two parts describe the activities of ontology evolution and versioning in order to clarify their respective roles and functions in an OCM process.

1.2 Ontology Evolution

Ontology Evolution is an area of research that attempts to address the problems of **updating the ontological knowledge and indirectly its reuse in a changing world**. An existing ontology should be modified over time to reflect changes happened in the real world, new user requirements, or correct errors or lacks in the original design (non optimal or poor conceptualization, etc..). These changes help to **incorporate new features** and ensure the **integration of incremental improvements**, but can **generate inconsistencies**.

1.2.1 Ontology Evolution Definitions

As defined above by (Flouris & al, 2008), ontology evolution as type of change, generally refers to the process of **modifying an ontology in response to a need of change in the domain or its conceptualization**. Applying changes to an ontology is normally used to make it more accurate and appropriate to the domain it describes and its purposes of use. However modifying ontology presents several difficulties both practical and theoretical. According to (Djedidi, 2009) it is not always easy to understand clearly what is expected of a need of change, nor how this change can be achieved. Also, according to (Luong, 2007), changes applied in a certain part of the ontology can also cause inconsistencies in other parts of the ontology, in dependent ontologies and also in applications using this modified ontology. They also may particularly affect semantic annotations using concepts or properties defined in this ontology. That is why the definition of ontology evolution tends to be enriched by the notion of consistency maintenance. (Jaziri, 2009) defines ontology evolution as a formal interpretation of change requests while maintaining the consistency of the ontology. A state of the art in the field confirms the tendency to enrich the definition of ontology evolution with the notion of **consistency maintenance**.

The evolution of ontology must follow a precise methodology to ensure the management of the impact of changes on the extension of the ontology but also to any dependent artefact (the ontology metadata, user applications, etc.). (Rogozan, 2009) extends the definition of (Jaziri, 2009) by stating that consistency maintenance should include the preservation of the integrity of all the elements related to the ontology.

1.2.2 Ontology Consistency Maintenance

Maintaining the consistency of an ontology aims at **ensuring compliance with the different constraints of the language** representing the ontology. According to (Rogozan, 2009), an ontology is considered consistent if all the constraints associated with the ontology are met. Before applying a change on ontology, the various constraints each change should meet to ensure a consistent application have to be studied. The main constraints are related to the **syntax** (model) or the **logic** (axioms) of the language. They respectively correspond to the **structural consistency** and **logical consistency** of the ontology. According to (Horrocks et al., 2003), maintaining structural consistency implies then **modelling respectful changes w.r.t. the ontology language constructs** (e.g. conforming use of the concept, of property, etc.). Structural consistency only cares about the ontology conformity to these structural constraints, while the logical consistency, addresses the question of whether the ontology is " semantically correct", i.e. contains no contradictory information. The meaning of logical consistency corresponds to the **satisfaction of the axioms of the ontology at the logical level**. In most work, respect of these two types of consistency is enough to qualify the ontology as consistent. Maintaining ontology consistency may also relate to any other constraint related to its use that the ontologist assess as necessary. (Djedidi, 2009) defines, for example two other types of consistency in her evolution framework: the conceptual consistency and the consistency of modelling. The first refers to the conceptualization of the ontology that is to say, the way it is modelled but also evolved. It corresponds to the constraints of good conceptualization she identified in her framework. The second compares the conceptualization of the domain modelled by the ontology, to see if it reflects the domain knowledge. However, structural and logical inconsistencies caused by misapplication of changes are the major problem of ontology evolution. If they are not managed, they can make the ontology unusable. The ontology will not be able to be requested anymore by the domain users who need to reuse its knowledge, even less for inference purposes. Knowledge reuse is indeed a major step in the development process of an ontology. A **good ontology evolution process** must be able to ensure the **maintenance of the structural and logical consistency at any time in the life cycle of the ontology**. This task cannot however be manual if the ontology is represented in an expressive formal language and counts a huge set of axioms. It would be illusive to manually manage the effects of changes in the ontology for each application of a change as the size and complexity of such ontologies can easily exceed human capacity. An automatic or semi-automatic consistency maintenance step is then necessary.

1.2.2.1 Structural Consistency Maintenance Approaches

According to (Haase & Stojanovic, 2005) structural consistency considers the constraints that are defined for the ontology model according to **constructs of the language allowed** to form the axioms of the ontology. (Haase & Stojanovic, 2005) focuses on the OWL ontology language, which considers different sub-languages (sometimes called species) such as OWL DL, OWL Lite and OWL Full. These sub-languages differ in terms of allowed constructs and can be defined in terms of constraints on these constructs. (Haase & Stojanovic, 2005) precises that the role of these sublanguages is to be able to define ontologies that are “easier to handle”, either on a syntactic level to for example allow easier parsing, or on a semantic level to trade some of the expressivity for decreased **reasoning complexity**. Consequently, when evolving an ontology, a domain expert should ensure that it does "not leave the sub-language". However, because of the variety of OWL sub-languages, it is not possible to work with a set of consistency conditions predefined and a fixed structure. To ensure the application of a change do not make the ontology leave its

sublanguage **expressivity**, the (Haase & Stojanovic, 2005)'s approach consists in the definition of sub-languages in terms of conditions of arbitrary structural consistency conditions (constructs not allowed) with the corresponding resolution functions.

A fixed structure with predefined consistency constraints is however a practical and safe solution to strictly limit the use of constructs of a unique language. An ontology model defined by a fixed structure of axiom subsets derived from its representation language constructs avoids adding axioms with wrong constructs and leaving the language expressivity. Based on this fixed structure, maintaining structural consistency simply amounts to **ensure that the structure of the axioms contained in each subset of the model is not affected** by the change to apply. Changes involving the deletion of ontological elements can cause such structural inconsistency. When an axiom is added to the ontology, i.e. in one of the model subsets, it depends on ontological elements such as concepts, datatypes, roles, attributes, instances or a data values that have been already added. The deletion of one of the elements on which the axiom depends, will turn the ontology structurally inconsistent. For example if a concept, described by an axiom specifying a concept disjunction, is deleted. A possible approach consists in defining structural consistency constraints for the language, in terms of axiom dependencies for each of the ontological element deletion changes defined in the related model. In many proposals of the literature, this issue is treated during the logical consistency maintenance step. This kind of inconsistency can actually be seen as a logical contradiction: a concept, which does not exist anymore, cannot be used anymore to build an axiom. Yet it does not relate to a semantic constraint but a structural one: an concept disjunction between concepts must at least assign two concepts. I chose to reuse this fixed approach to validate the **fourth criterion** of my thesis.

1.2.2.2 Logical Consistency Maintenance Approaches

Logical consistency refers to the formal semantics of the ontology and to its **satisfiability** in the sense that it is semantically correct and does not have any logical contradiction (Haase & Stojanovic, 2005). An automated logical consistency maintenance process involves the respective tasks of **diagnosis and repair**: first, the cause (or a set of potential causes) of the incompatibility must be determined, and, second, it has to be repaired. The literature presents different approaches for maintaining logical consistency generally based on **corrective** and/or **preventive resolution approaches**. The corrective approach is the simplest. It repairs inconsistencies of the ontology when a reasoner detects them. The preventive approach occurs before applying the changes to the ontology. It defines a set of pre-conditions for each change, forming a model of consistency of the ontology to check before applying the change. (Haase & al, 2005) presents a study on the management of ontology consistency in ontology evolution, which summarizes these two approaches.

According to (Haase & al, 2005) corrective resolution of inconsistencies consists of two tasks: research and correction of inconsistencies. The detection of the source of the inconsistency is the first step of the approach. Authors define the source of inconsistency as a set of axioms whose joint presence in the ontology model makes the ontology inconsistent. The idea is to find the **smallest set of inconsistent axioms** of the ontology. Once the source of inconsistency is found, the existing conflict in the identified set of axioms remains unresolved. This task is difficult, because in most cases, there is not only one way to solve a conflict, but a set of alternatives. Often, there is no logical criterion that can be used to orient the selection of the best resolution. A common approach is then to **allow the user to resolve the conflict once it has been located**.

According to (Haase & al, 2005), preventive resolution approaches generally define a **semantic model of changes** to ensure the logical consistency of an ontology. Depending on the expressivity level of the language, a higher or lower number of changes can be taken into account. The various constructs of the language have to be firstly studied to **derive logical constraints**. The second step is to **identify potential inconsistencies caused by changes to apply**. The third step is to resolve these inconsistencies. **Belief Change** research area has inspired relevant works dealing with ontology

preventive inconsistency resolution (cf. (Flouris & al, 2005, Ribeiro & Wassermann, 2009a, Haase & Stojanovic 2005)). (Flouris, 2006) identifies two relevant types of belief change that can be transposed to recover the ontology consistency : revision and contraction. **Belief revision** consists in generating additional changes for a transition to another consistent state (Ribeiro & Wasserman, 2009b). The strategy can be summarized as follows: when an ontology becomes inconsistent, because of the application of a change, a resolving change operation is generated and its application on the inconsistent ontology restores the consistency of ontology. **Belief contraction** consists on the contrary in removing inconsistent changes (Ribeiro & al, 2009). It can rely on the **monotonicity property** of the ontology language considered. The monotonicity property states that if the ontology language is monotone (this is the case for description logics, such as $\mathcal{SHOIN}(\mathcal{D})$, and formal languages based on them, such as OWL DL), and **if the ontology is satisfiable, it will always be satisfiable whatever the axiom deleted**. This means that, if the evolved ontology is inconsistent, some of its axioms must be removed to restore its consistency state, while taking into account its evolution. The difficulty lies in the selection of axioms to be removed because the impact on the ontology should be minimal so that the evolution is done correctly. Belief change has inspired (Haase & Stojanovic, 2005)'s logical consistency preventive resolution proposal. It has also inspired the approach, called CLOCk presented in (Gueffaz & al, 2012) using a model checker analysis to identify minimal inconsistent sets of changes to revise or delete from the ontology log of changes. This approach has been transposed and formalized in this thesis to validate the **third criterion** of my thesis.

1.3 Ontology Versioning:

The **Ontology Versioning** research area is very close to the Ontology Evolution one so that they are sometimes confounded. It addresses the issue related to the **management of the different versions of an ontology**, namely those generated by the ontology evolutions. The ontology versioning activity envisions many features like: accessing a certain ontology version, navigating between the ontology versions or comparing them to determine their compatibility. These features are managed by a **versioning system**, which can be based on the ontology identifiers or on a history of changes also called log.

1.3.1 Ontology Versioning : Definitions

According to (Flouris & al, 2008), the purpose of versioning is to **give a transparent access** to the different versions of an ontology. It supports the different versions of an ontology as input and produces an ontology versioning system to manage them. It uses ontology identifiers (IRI) to **identify versions**, provides a transparent access to the desired version and determines the compatibility between each of them. In the field of OCM, the definition of versioning is extended. It aims at managing the multiple versions of the ontology, which are not necessarily **chronological** (c.f. multi-temporal versioning (Grandi, 2009). They can be both **local** and **online, collected, distributed** (e.g. semantic search engines : (Allocca & al, 2009). In these cases, the versioning activity is **retroactive** for all versions of the ontology already exists. However, to meet my thesis fifth criterion, my proposal has to integrate a versioning system capable of managing versions of the ontology since its creation. This can be defined as a dynamic or an **incremental versioning**, which, when an evolution of the ontology is validated, takes into account this new version. Incremental versioning should not be confused with retroactive versioning, which can be found in most of the proposals. Incremental versioning provides a new feature to versioning systems: being able to **generate a new version of an evolved ontology**. The incremental versioning process can be described as follows: when an ontologist has finished adding changes on an ontology, he validates it and the versioning system generates and records a new version of the ontology. This new version can replace the former one or simply coexists with it. This point will help meet the **fifth criterion** of my thesis.

Additionally, (Allocca & al, 2009) attributes to versioning the functions of tracing the evolution of an ontology according to its versions, querying them, accessing the desired version. Many studies are limited to the use of IRIs of the ontology versions to perform these operations such as (Maynard & al, 2007). However, this identifier is not always known. In addition it does not allow finding a version of the ontology based on a criterion other than the IRI. It may be particularly interesting to access a version in which some changes occurred during the evolution but whose IRI is not known. Tracing the changes applied between two versions also allows to answer this problem.

1.3.2 Representing and Tracing Changes for Versioning Purposes

Different approaches are proposed in the literature to **represent** and **trace changes** in ontologies. Among them, can be noticed history of changes, called versioning logs or evolution logs, and comparison methods representing the conceptual differences between each version or sets of transformation. Most of works focuses on the use of logs for the representation of changes. According to (Klein & Fensel, 2001) the representation of a change in a **log**, or other support for tracing changes, should contain certain information. The main information is the specification of the changes in the form of **ontological operations applied to the elements** of the ontology. The second one is the conceptual relationship between the former and the new ontology version, i.e. a mapping between the concepts of the one and the concepts of the other. Ideally, there may also be **metadata about the change** and **information about the consequences of each change** in terms of gain or loss of ontological elements in the ontology. According to the study of (Yildiz, 2006), **versioning logs** record the different versions of an ontology by defining each ontological element at a given time. For each concept, instance, role and attribute created in the ontology, a new occurrence of an "EvolutionConcept" concept is created in the versioning log. This occurrence is defined by a transaction time, a state (confirmed or pending), an identifier and a cause. However, recording the exhaustive description of each ontology version may produce logs that are bigger than the ontology itself. It can cause problems for their storage if the ontology is large and complex and if it often evolves. Unlike versioning logs, an **evolution log** does not exhaustively describe the different versions of ontologies. According to (Liang, 2006), an evolution log is a file that only **lists the interpretations of the changes** applied on an ontology. For each list of changes applied, the ontology is updated and these changes are stored in the evolution log. This log will be used later for querying a certain version of the ontology. An evolution log can then be seen as a history file of the different changes that have affected a given ontology. Another approach for change tracing, is the **representation of the difference between two versions** of an ontology. It uses a comparison of the two terminologies. This difference can be accessed using a process, "diff", which returns the list of modified axioms between the compared versions as formalized in (Noy & Musen, 2002). The representation of the difference can wear the form of a conceptual change, specifying the conceptual relationship between the old and new ontological structures. This solution involves defining external operators between ontology versions. Finally the difference between two versions can be represented in a **set of transformations** containing external operations defining the transformation of the ontology from one version to another. However, all the approaches based on the representation of the difference are not appropriate for incremental versioning as it takes at least two existing versions of an ontology as input. So they are better suited for retroactive versioning.

A convenient approach, to **retrieve and navigate between ontology versions** allows the user selecting search criteria he has himself defined. These criteria can be the state of an ontological element, its last modification, a list of operations applied on it, etc. Approaches using evolution logs, or a concept of change that defines transformations from operations of evolution can easily identify the desired version of an ontology according to this kind of criteria. Other criteria such as contextual information would be useful to facilitate the search for the user. Contextual information can be an ontology version name, its title, its version number, its description, the reason for the change, the expected consequences of the

changes applied, its creation date, etc. Information time can be saved automatically, such as modification date, time, etc. These metadata can be easily represented in logs.

To meet the **fifth criterion** of my thesis, the use of evolution logs appears to be the optimal solution for the representation of changes and ontology version retrieval. Also, to be incremental, sufficiently precise and to give a transparent access to the desired ontology version, ontology versioning must be based on the evolution iterations of the ontology. It is from this observation that ontology change management makes sense.

1.4 Ontology Change Management

This part presents the **Ontology Change Management** (OCM) activity, which merge Ontology Evolution and Versioning to manage ontology changes in a **unified methodology**.

1.4.1 Klein and Stojanovic's Methodologies

In his PhD thesis (Klein, 2004), Klein defines Ontology Change Management (OCM) as the **fusion of Ontology Evolution and Versioning**, in the perspective of **ensuring the suitability of ontologies with changes**. He proposes a first ontology versioning management methodology based on this principle (c.f. (Klein et al., 2002) (Noy & Klein, 2004) (Klein & Noy, 2003)). He designs an **incremental versioning model** for storing and accessing the different versions of an ontology generated during its development. He recommends to model ontology versions relationships, to reflect changes made between versions and clarify the semantic relations between ontological entities of two ontology versions. He also advises to add metadata about changes, such as their duration. His approach provides an analysis of the relationships between the versions of the ontology model, but it does not suggest, however, how the versions of the ontology can be created. It does not either address the management of access to dependent artefacts (referenced resources, other ontologies or applications).

Stojanovic, Klein's contemporary, focuses, in her PhD thesis (Stojanovic, 2004), on the ontology evolution aspect by proposing the first ontology change management methodology integrating the vision of Klein. In (Stojanovic, 2002), she first stresses all the conditions to be met by an ontology evolution management system, such as consistency maintenance and propagation of changes to dependant artefacts of the ontology. An **ontology evolution process** is derived in six phases to guide the consistent application of changes: **change detection, change representation, change semantics, change implementation, change propagation** and **change validation**. Change detection and change representation phases respectively allow to detect and to represent the changes in a suitable format. A **taxonomy of fine-grained ontological changes** is derived from the KAON language (Bozsak & al, 2002) that can be performed in the course of ontology evolution. However this granularity of ontology evolution changes is not always appropriate because some change intents may imply the application of successive fine-grained changes to bring the ontology into desired state. To avoid process error prone and the application of unnecessary changes, the author recommends the formal definition of **composed changes** that represent a **group of basic changes applied together**. For the change semantics she advises the use of a richer description of the changes in order to determine the semantic role of ontology entities. In order to avoid performing undesired changes, before applying a change to the ontology, a list of all implications to the ontology can be generated and presented to the user. She defines a **preventive** and **a corrective resolution approach** to handle **logical consistency maintenance**. When the user approves the changes, those can be performed by successively resolving changes applied on the terminology from the list during the change implementation phase. According to her, if changes are cancelled, the ontology should remain intact. To preserve the consistency of the ontology with the instances and with dependant ontologies, she provides a change propagation phase. The changes are thus implemented on the assertional level of the ontology after they are validated on the terminological one. Concerning dependant ontologies that reuse or extend the ontology, the ontology update might corrupt them and consequently all artefacts that are

based on these ontologies. The author states that the evolution process should recognize which change in the ontology can affect the functionality of dependent applications in order to react correspondingly. Finally, the user can commit the implemented and propagated changes at the change validation.

Both Klein's and Stojanovic's methodologies have inspired most of the ulterior OCM proposals in the literature, which have permitted to apply the OCM principle on more specific ontological issues. Also, Klein's and Stojanovic's OCM methodologies can be qualified as **formal** for several reasons. Each ontology evolution iteration follows an analysis phase of the need of change, which stands as an ontological analysis necessary for formal ontologies. Changes to be applied to ontology are tailored from the constructs of the ontology representation language. Representation languages considered in both methodologies, respectively KAON, which is a similar but less rich than OWL DL, for Stojanovic, and OWL, including OWL DL, for Klein, are formal languages. Thus changes are modelled and formally represented, their impacts are assessed, structural and logical consistency of their application on the ontology are qualified before any validation. This is why, the proposal presented in my thesis has been mostly inspired by these two formal methodologies to **meet the first criterion** of my thesis.

1.4.2 Related Works

Other methodologies have been inspired and have come later complete these OCM methodologies. Six of them are described and compared below.

(Djedidi & Aufaure, 2008) describes an approach for evolving and enriching OWL ontologies maintaining their consistency according to a quality model. The authors define a process change management conducted through four phases: detection of inconsistencies, resolution alternative propositions, quality assessment and application, and final validation of the changes. The change validation phase takes into account both logical consistency qualification of the change application and quality assessment of the evolved ontology. A quality model is defined and applied to assess the impact of the proposed resolution alternatives on the quality and to guide the resolution of inconsistencies while minimizing dependence on the ontologist. The alternative preserving the quality is automatically selected. It consists of additional changes to be applied in order to maintain the consistency of the ontology and its quality.

(Jaziri, 2009) presents a general methodology developed for monitoring the evolution of the ontology by the creation of a new version suited to changes occurring in the domain. The author studies the types of changes, which may be involved and their conceptual and semantic consequences on the ontology. He also presents a taxonomy of basic and composed changes. These changes are usual ones and are not derived from a particular language because the methodology aims to be general and applicable on ontologies whatever their specification language. For each type of change a change kit, composed of the potential inconsistencies that can be caused by this change and the corresponding resolution alternative, is provided. This allows assessing logical consistency of the application of changes and resolving it. A validation phase produces a new version of the ontology, which updates the original ontology.

(Rogozan, 2009) presents a methodology describing the complete ontology evolution process including distributed ontology version management on the Semantic Web. This process consists of eight phases among those two are major ones: the evolution phase and the implementation of the evolution phase. The author also defines an ontology describing changes applicable to OWL-DL ontologies. This ontology extends the change taxonmy of (Klein, 2004), by adding a number of features such as: a typology of complex changes, a description of the effects of changes on a semantic search engine optimization (SEO), a clarification of the terminology used to characterize the complexity of change. She proposes a model of change logging and a representation language for changes. The author has developed a method and a tool to analyse the effects of changes on the semantic SEO resources and propagate them to maintain access and interpretation of the referenced resources.

(Luong, 2007) presents a new approach to managing the evolution of an Enterprise Semantic Web (ESW) through a new system called COSWEM. This approach focuses on the resolution of inconsistencies

in semantic annotations caused by the evolution of an ontology on which they are based. It also allows the user comparing the differences between two versions of the ontology by viewing the changes between them and the ontological elements impacted by these changes. It also determines the impact on the semantic annotations based on the ontology thanks to an evolution log tracing the changes. CoSWEM system handles two major tasks: detection of inconsistencies generated on semantic annotations and correction of the inconsistent semantic annotations. These tasks aim at ensuring the overall consistency of an ESW. They are performed automatically or semi-automatically with the assistance of the user, which can be an ontologist or an annotator. A classification of 26 ontological changes is defined according to the following criteria: the abstraction level (simple or composed changes), the assignment of the annotation (mandatory or optional corrections) and the affected ontological elements (concept or property). These changes are formally represented in an evolution log that also serves ontology evolution. As (Stojanovic, 2004) it offers a preventive and a corrective approach to manage the consistency of semantic annotations. The first approach can only be applied if a log of changes in the ontology has been preserved. The second uses, in the absence of an evolution log, a rule-based detection and correction of inconsistencies. Resolving the annotation consistency can then be proceeded by reusing the consistency resolution strategies already implemented on the ontology.

The approach presented in (Eder & Koncilia, 2004) includes a formalism representing ontology versioning as directed graphs. It is based on the concepts and techniques developed for temporal databases, schema evolution and database versioning. A graph includes all the ontology versions by incorporating the concept of validity period in the definition of concepts and relations composing the graph. A version is produced by operators that add, delete or update the graph.

(Plessers, 2006) proposes a change management methodology for semantically heterogeneous ontologies representing different views of a domain in a distributed environment. The methodology embeds an ontology evolution process that allows ontology engineers to request changes for the ontologies they manage; ensure that the ontology evolves from one consistent state into another consistent state; guarantee that the depending artefacts of an ontology remain consistent after changes have been applied; provide a detailed overview of the changes that occur. A versioning log stores for each concept ever defined in an ontology the different versions it passes through during its life cycle. The Change Definition language, used to represent changes is a temporal logic based language that allows ontology engineers to formally define changes. The purpose of the change definitions expressed in this Change Definition Language are twofold: they are used for both requesting and implementing changes to an ontology, as well as detecting occurrences of change definitions in the evolution of an ontology.

1.5 Discussion

This part presents a discussion on OCM related works. The eight methodologies presented in previous part are firstly compared in Table 4, according to the eight criteria defined for the proposal of my thesis. It allows to identify the criteria still unsatisfied among these eight ones.

According to Table 4, the OCM methodologies of the seven proposals all propose a **formal methodology**, which at least allow to apply formal changes and maintain the ontology consistency and expressivity level. Most of them have chosen a formal representation language expressive enough to specify formal ontologies. So the 1st and 2nd criteria are satisfied. They especially addressed the logical and structural consistency issues related to the application of changes on an ontology, which respectively constitute the 4th and the 5th criteria of my thesis. However, four criteria remain unsatisfied or partially satisfied by the existing methodologies: 3rd, 6th, 7th and 8th. None of the proposals handle change management on formal ontologies representing the several views of a domain (3rd criterion). The closest proposal is (Plessers, 2006)'s approach, whose issue is the change management of semantically heterogeneous ontologies representing different views of a domain in a distributed environment.

However the propagation of changes is not addressed between these ontologies but from each ontology to their dependent artefacts (applications, reusing ontologies). Evolution and versioning tasks are only done on the ontology in most of the proposals, and none proposes their use on the different views of this ontology (6th criterion). Most solutions for the change propagation phase focus on the impact resolution of changes on the ontology instances, on the dependant ontologies and artefacts such as applications, semantic annotations, semantic SEO. None of them addresses the propagation of changes from the ontology to its views (7th criterion). The adaptation of an ontology to new points of view is not addressed either (8th criterion).

This state of the art highlights the following observations. **OCM is performed over time on an ontology that chronologically evolves in its entirety**. It is mainly used for the update of the ontology to correct its conceptualization, when its described field has evolved, or when errors or potential improvements are identified. It does not address the **evolution and versioning management of the specific assumptions users might have on the domain**. It has not therefore been thought to manage formal ontology as defined in the previous chapter, i.e. an ontology representing all the assumptions of users on the domain. It is not intended to guide the specialization of an ontology to a sub-domain that it describes. The propagation of changes from an ontology to various specialized views is thus not addressed either. The **specification of user views on an ontology** is the subject of study of the **Ontological Views** research field. Therefore, to understand the solutions meeting the remaining four criteria, a state of the art on Ontological Views is presented in the next section.

	Criterion	Stojanovic	Klein	Luong	Jaziri	Djedidi	Rogozan	Eder	Plessers
1	Formal Evolution Process with: Change Analysis + Representation +Consistency Maintenance + Propagation	Change Detection +Representation +Semantics +Implementation + Propagation +Validation	Change Detection +Change Propagation +Change Representation +Change Implementation +Version Compatibility + Version Alignment	Change Representation +Consistency Maintenance +Propagation	Change Representation +Consistency Maintenance +Validation	Change Representation +Consistency Maintenance +Change Implementation	Change Detection + Representation +Consistency Maintenance + Validation + Implementation + Validation	Change Representation + Validation	Change Detection + Representation + Propagation + Validation
2	DL $\mathcal{SHOIN}(\mathcal{D})$ or OWL DL Language	KAON	OWL (including OWL DL)	RDF(S)	Language undefined	OWL DL	OWL DL	Oriented Graph	OWL DL
3	Formal Ontology representing the several views on the Domain	N.C.	Distributed Ontologies	Ontologies Used for Semantic Annotations	Any Ontology Types Representable in UML	Local Domain Ontologies	Web Semantic Referenced Ontologies	Temporal Ontologies	Semantically Heterogeneous Ontologies of the same Domain in Distributed Environment
4	DL $\mathcal{SHOIN}(\mathcal{D})$ or OWL DL Logical Consistency Maintenance:	KAON & OWL Lite Logical Consistency Maintenance	N.C.	RDF(S) Logical Consistency Maintenance	Language Independent Logical Consistency Maintenance	OWL DL Logical Consistency Maintenance	N.C.	N.C.	OWL DL Logical Consistency Maintenance
5	DL $\mathcal{SHOIN}(\mathcal{D})$ or OWL DL Structural Consistency Maintenance	KAON Structural Consistency Maintenance	N.C.	RDF(S) Structural Consistency Maintenance	Language Independent Structural Consistency Maintenance	OWL DL Structural Consistency Maintenance	N.C	N.C.	OWL DL Logical Consistency Maintenance
6	Evolution/ Versioning of Ontologies and its user assumptions	Evolution/Versioning of Ontology	Evolution/Versioning of Ontology	Evolution/Versioning of Ontology & Semantic Annotations	Evolution/Versioning of Ontology	Evolution/Versioning of Ontology	Evolution/Versioning of Ontology and its Semantic SEO	Evolution/Versioning of Ontology	Evolution/Versioning of Distributed Ontologies
7	Change Propagation: Ontology to its Views and Views to Ontology	From ontology to instances, dependant ontologies & artefacts	N.C.	From Ontology to Semantic Annotations	From Ontology Concepts to Ontology Sub-Concepts	N.C.	From Ontology to SEO	N.C.	From Ontologies To Dependant Artefacts
8	Adaptation of Ontology to new Points of View	No Adaptation	No Adaptation	No Adaptation	No Adaptation	No Adaptation	No Adaptation	N.C.	N.C.

Table 4. Comparison of OCM proposals w.r.t. my Thesis 8 criteria

2 Ontology Views

This second section introduces in a first part the **Ontological Views** research field covering a broad spectrum of solutions to the specification and management of ontology views. The idea here is to identify, in the existing view specification methodologies, the appropriate approaches, which could satisfy the four remaining criteria for the conception of my proposal: 3rd, 6th, 7th and 8th. The 3rd criterion is the representation of the several user assumptions on the domain in a formal ontology. The 6th is the evolution and versioning management of these views according to the ontology evolution. The 7th is the propagation of changes applied from the ontology and to its views and from the views to the ontology. The 8th is the ability to adapt the ontology to new user assumptions. A second part presents the different implementations and uses through the existing view specification methodologies and highlights the issues of these proposals in the context of OCM. A third part consists in a discussion identifying the advantages and lacks of views according to the four criteria.

2.1 Ontology Views in the Literature

This first part deals with the motivations for the introduction of views in ontologies, their objectives, and their definition.

Since 2007, many works have combined ontology evolution and versioning into ontology change management methodologies derived from Klein and Stojanovic's methodologies. These proposals constitute a consequent background for ontology change management but they do not take into account certain specificities of ontologies. One of them is the fact that **ontologies are decentralized data** (Rajugan & al, 2006). It means that multiple versions of the same ontology evolution are bound to exist over the Web and must be supported. It implies that ontology chronological evolution is not enough to manage change in ontologies. Actually, **managing different parallel versions of a same ontology** would bridge this gap. Another unaddressed known characteristic is that ontologies are meant to grow during their lifecycle and may **become too large to be used in their original scale** by potential applications. Indeed, ontology development implies a dynamic and incremental process starting from the **creation of a brute ontology, which has to be revised and refined** (Djedidi, 2009). Refinement often leads to the **improvement of the ontology level of detail** corresponding to the addition of new elements to its conceptualization. Therefore the ontology size may increase after each refinement process, with no guaranty that the ontology is still manageable by applications and understandable by humans. In the literature, ontology views have been defined to bridge this ontology size issue and **improve ontology reusability**.

Reference ontologies, such as the University of Washington's Foundational Model of Anatomy (c.f. FMA in (Rosse & Mejino, 2003)), are intended to support the domain knowledge requirements of multiple disparate applications**.** They are often too large or too complex, however, for any specific application (Detwiler & al, 2010). The problem is the same for **large domain ontologies** reused by specialized business applications. This is the case of biomedical domain ontologies like Gene Ontology (Ashburner & al, 2000) or CheBi Ontology (Degtyarenko & al, 2008). Automated or semi-automated evolution of these large ontologies is often impossible to realize because of their complexity. In addition, the "world view" provided by such domain ontologies may not match exactly the views required by particular applications. According to (Noy & Musen, 2004), **potential applications will not access to the entire ontology**. Access to the whole ontology can be unnecessary and slow if the ontology is complex. In order to utilize these ontologies, applications often require **custom ontology views tailored for use within their specific context**. In this case, views can optimize the access time and query processing of ontology by only loading a small portion of this ontology. In recent years, many academic and industrial

research directions have focused on these issues and some of the notable works include (Volz & al, 2003, Do & Rahm, 2004, Hadzic & Chang, 2004, Ceravolo et al, 2006, Wouters, 2006). Displaying portions of ontologies is then crucial to allow OCM supporting large ontologies evolution. Also, in user-centred applications, users usually need to use only a small portion of their resources or may **not have the right to access certain parts of the ontology**. Views are used then to manage access policies, profile, context and data security for users (Rajugan & al, 2006). Moreover, adaptation of the ontology to several contexts or uses is one of the change management objectives. Finally, the **overall understanding of a complex ontology by the community may be impossible** (Rajugan & al, 2006). A human cannot then realize evolution of such ontologies.

To resolve these issues, views should produce an **understandable and manageable portion of an ontology for local applications and users** as a means of enabling them to use standard well-developed ontologies (Orbst & al, 2003). This requirement groups all the objectives of views cited above. This is the unique guideline we will retain all along the section because it quite well fits with the four remaining criteria: it allows to **adapt the ontology to a new user assumption** by producing a understandable user portion of the ontology (8th criterion); this portion must be manageable, so it has to be **evolvable and versionable** (6th & 7th criterion); different views can be produced from to **represent all the user assumptions** on the described domain (3rd criterion). Several definitions and implementations of ontology views have been studied in the Ontology View Management specific research field. However no agreement was found. Nevertheless, a view generally is a **subset specification on an ontology**, **which allows extracting a manageable portion of the ontology capable to be used and queried by applications like the whole ontology**. The resulting subset can be generated not only as a sub-graph of the ontology but also as an independent ontology, itself being a new interpretation of the domain. It can be considered as a new parallel version of the actual ontology validating the decentralized quality of ontologies.

2.2 View Specifications Approaches

Two main approaches exist in the literature to deal with ontology views and are presented below: **query language based approaches,** which use queries to select sub-graphs of the ontology, and **subset extraction approaches**, which uses subset extraction techniques to provide sub-portions of the ontology. Other view specifications approaches based on rules, pipes and named graphs are also exposed.

2.2.1 Query Language Approaches

In databases, a **view is specified as a query**: any instances satisfying this request constitute the view. Researchers from the database field have been working on using views directly responding to requests. Since the views are themselves queries, this field of research reformulated the user query result to express it as physical and searchable data in a database table: the case of **materialized views**. So it is within the same perspective that ontology views defined by queries are specified on an ontology. Ontology query languages are used and extended for this purpose and provide several features to manage and update the views.

Until now, several different RDF query languages have been proposed for querying within the semantic web, including RQL (Karvounarakis & al, 2002), RDQL[7] and SPARQL[8]. A number of proposals have been made for extending SPARQL's functionality, including SPARQLeR (Kochut & Janik, 2007), ARQ[9],

[7] *RDQL: Query Language for RDF: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/*
[8] *SPARQL: Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/*
[9] *ARQ: SPARQL Processor for Jena: https://jena.apache.org/documentation/query/*

SPARQL++ (Polleres & al, 2007), CPSPARQL (Alkhateeb et al.), nSPARQL (Perez et al.), NetworkedGraphs (Schenk & Staab, 2008) and vSPARQL (Shaw & al, 2011). Below are detailed RQL, RVL and vSPARQL proposals.

(Volz & al, 2003) defines a language for view based on the query language RQL. RQL is a declarative query language for RDF that allows querying over both resource descriptions (instances) and schemas (terminology). In this framework, a view represents and is **defined as a new concept or a new role in the ontology**. The author restricts RQL queries that can be used in order to ensure that the view returns only unary or binary relations (respectively instances or assertions). The definition of **views on concepts** involves two components: (a) users must define an arbitrary RQL query, which returns unary tuples of resources - these tuples constitute the instances that are in the extension of the view-, (b) the view must be properly classified in the concept subsumption hierarchy. **Views on roles** can be defined using arbitrary queries, which return binary tuples. Besides the query itself, additional information is required to define a view on roles: (a) the definition of the domains and ranges of the view, (b) embedding the view into the role subsumption hierarchy and (c) forcing the query to return binary tuples. Therefore, RQL **expands the RDF-S graph's concept and role subsumption hierarchies**. Users can also perform a sub-graph extraction over these hierarchies.

(Magkanaraki & al, 2003) have chosen to deepen this approach to define views based on queries. RVL is a view definition language for creating virtual resource descriptions and schemas; RVL uses RQL as its query language. In RVL, mechanisms to **restructure the original concept and role subsumption hierarchies** are proposed, enabling the creation of new instances, concepts or roles. From there, the view definition includes not only the result of the new structuration itself, but also **a set of predicates that define these new structures**, linking them to the query results. RVL and RDF-S lightweight ontologies (Volz & al, 2003) both leverage RQL to provide **declarative mechanisms** for defining views over ontologies. The RDF-S semantics of the underlying data are used to ensure that the defined views' semantics match the data, so that new concepts and roles can be defined in the view. Also, RVL permits the definition of views over **different namespaces**, making possible **multiple intermediate sub-views** within a single view definition file.

Proposed as a query based view specification approach by (Polleres & al, 2007), SPARQL++ is an extended version of SPARQL designed to allow **mapping between RDF vocabularies**; the extensions include nested queries, external functions, and aggregates. Extended graphs allow data to be combined with "CONSTRUCT" statements to **add new triples**.

NetworkedGraphs, recommended by (Schenk & Staab, 2008), allows users to define RDF graphs by combining explicitly listed data and views over other RDF graphs, specified via SPARQL "CONSTRUCT" statements. Views are dynamically evaluated by querying remote SPARQL servers holding the necessary graphs. The remote graphs may themselves be defined by views over data at other machines, thus creating a **network of graphs**. To ensure decidability when querying these graphs, NetworkedGraphs **prohibits value creation via blank nodes** in "CONSTRUCT" templates.

In (Shaw et al., 2011) is presented a general solution for RDF information set reuse inspired by database views in order to enhance the **reusability of views specified by queries**. The view definition language, vSPARQL, allows applications to **specify the exact content that they are interested in and how that content should be restructured or modified**. A query manager tool has been constructed in order to allow view queries to be edited, executed, and stored for reuse. Applications can then access relevant content by querying against these view definitions. The query manager also supports the **storage of materialized view results** upon which further queries may be issued. This view definition language allows intermediate results such as (Noy & Musen, 2004)'s derivations to be conditionally combined or intersected. It allows the user to specify exactly which axioms are relevant and how those axioms should be arranged or augmented. Even when those axioms are modified in the view, subsequent queries can still be answered against the original ontology, ensuring that information set updates are reflected in the query results.

2.2.2 Manual, Structural and Logical Approaches

Parallely to query language approaches, three approach types, which, like vSPARQL query approach, provide materialization of a subset of an ontology for reuse, have been proposed: **manual, structural, and logical**.

Traditionally, **manual techniques** have been used to **derive a relevant subset of an information set** for an application's use thanks to a **partial mapping or other mapping techniques**. A developer acquires a copy of the information set, which can be called "**sub-ontology**" and then manually deletes, modifies, and adds axioms until the application's requirements are met. This definition of view as extraction of a sub-ontology is different from the vision of databases here because the view contains the knowledge of the sources selected by the ontology mapping, whereas query approaches generally only materialize a query as a new concept of the ontology. Below are detailed the main relative works of the literature. (Orbst & al, 2003) suggests an approach creating application views of ontologies to allow applications using standard ontologies without questioning the entire ontology. This approach combines views or "**contexts**" with **mappings between concepts in the views and in the ontology**. A mapping is generally used to associate each element of a given with one or more elements of a second one without impacting them. By extension it can be used to specify the relations between an ontology and its sub-ontology. Because it has no direct impact on them, it preserves the ontology and its extension. So the issue of consistency resolution does not arise before further evolution of the view, if the source ontology is known as consistent.

Structural techniques such as PROMPT's Traversal Views in (Noy & Musen, 2004) and Web ontology segmentation (Seidenberg & Rector, 2006) require a set of concepts and roles to be specified for inclusion in the result, which is a **self-contained subset of the ontology**. Starting from these **key elements** also called **starter elements**, the output set is grown by recursively adding the specified roles and concepts until a fixed point is reached. PROMPT's Traversals Views (Noy & Musen, 2004) allow **multiple distinct derivations to be united to produce a materialized result**. Also it introduces the notion of **view traversal** made from a semi-automatic mapping. A traversal is defined as the exploration of a sub-graph of the ontology from a concept departure point and of a given distance. It helps the user to explore a new ontology to **find the subset that covers a particular topic**. In a view traversal, a user **specifies a subset of the ontology to include in the view** by specifying concepts to include departures, roles and signatures concerned with these concepts and the maximum distance to cross from the departure (**depth of transitive closure**). This mechanism allows users to **extract portions of an ontology related to a particular concept or set of concepts**.

In contrast, **logical techniques** (Kontchatkov & al, 2009, Grau & al, 2007, Grau & al, 2008) have been developed for **deriving modules** from OWL-DL ontologies. A **signature is specified containing all of the key concepts that should be contained within a selected subset** of the ontology; leveraging the high-level semantics of OWL-DL, the extractor **grows the set of concepts and axioms** needed in the output module. For concepts in the signature, the ontology module is guaranteed to capture the meaning of the concepts, such that an application, which performs reasoning after importing the module, would give the exact same results as an application that performs reasoning after importing the entire ontology. Ontology module extractors are a powerful tool for extracting logically equivalent modules from an OWL-DL ontology. Logically equivalence means here that **a same query will return the same results** whether it is applied on the ontology or the corresponding module. However, they are not suitable for all applications' needs. The current approaches are limited to DLs such as $\mathcal{SHOIN}(\mathcal{D})$ and their derivative ontology languages such as OWL-DL.

2.2.3 Other Approaches: Rules, Pipes and NamedGraphs

Rules, pipes and **named graphs** are also used in other approaches for manipulating or transforming views.

In logic, a rule is the act of drawing a conclusion based on the form of premises interpreted as a function which takes premises, analyses their syntax, and returns a conclusion (or conclusions). **Rules** are mainly based on subsets of the First Order Logic and other extensions. Several different rule languages like TRIPLE (Miklos & al, 2003) and SWRL[10] have been developed for the Semantic Web. General rule languages, such as Jena's general rule language[11], can be used to derive new assertions without SWRL's semantic restrictions. Rule languages allow **new assertions to be inferred from existing ones** in an information set of the ontology that can be stored in a view. They also offer more expressiveness (e.g. construct for composite properties) than ontological query languages with efficient reasoning support. However, based on logic programming, they face the difficulties of decidability.

Several projects are building upon the notion of **pipes** to provide easy-to-use **mechanisms for transforming and aggregating RDF data in mash-ups**. A mash-up is a web application, which uses and combines data, presentation or functionality from two or more sources to create new services. Semantic Web pipes (Le-Phuoc & al, 2009) are powerful data level mash-up tools based on RDF. They are defined in XML and when executed they fetch RDF graphs on the Web, operate on them, and produce an RDF output. Operators are provided for extracting data from Web content, performing SPARQL queries over data, and inference. Similarly, SPARQLMotion[12] provides a GUI editor for pipelining RDF data sources and transformation operations together. Banach[13] also provides a set of operators that can be pipelined inside Sesame[14] to transform RDF data. (Westerski, 2006) proposes splitting mash-up development into two separate pieces – data-level and service-level – to simplify mash-up creation for users.

NRL (Sintek & al, 2007) uses **named graphs** and views in the context of the Social Semantic Desktop (c.f. SSD in (Decker & Frank, 2004)). SSD is a project to enrich and interconnect data from different desktop applications using semantic metadata stored as RDF triples. Named graphs are a key concept of the Semantic Web architecture in which a **set of RDF triples are identified using a IRI** allowing descriptions to be made of that set of triples such as context, provenance information or other such metadata. They are a simple extension of the RDF data model through which graphs can be created. But the model lacks an effective means of distinguishing between them once published on the Web. Graph views are used to **define specialized semantics and assumptions over RDF named graphs**. Graph roles declaratively assign meaning to named graphs. However graph views, like query and rule based views, are procedural, requiring the specification of what needs to be used to generate an output graph from an input graph.

2.3 Discussion

The OCM requirement for ontology views given in the previous section, namely **producing a understandable and manageable portion of an ontology for local applications and users**, seems to almost be reached by the different approaches presented. Some technical and functional issues, depending on the type of specification, still hinder the emergence of an optimal specification for ontology view for ontology change management. In this part we focus on subset extraction and query-based approaches, which are the main techniques used today and for which feedbacks have arisen.

First of all, the **extraction of a sub-graph of an ontology** is partially realized with the use of query languages. Indeed some problems arise with the use of IRIs when dealing with **ontology blank**

[10] SWRL: Semantic Web Rule Language: http://www.w3.org/Submission/SWRL/
[11] Jena: Semantic Web Framework for Java: http://jena.sourceforge.net.
[12] SPARQLMotion: http://www.topquadrant.com/products/SPARQLMotion.html.
[13] Banach: http://simile.mit.edu/wiki/Banach.
[14] Sesame is an open-source framework for querying and analyzing RDF data which contains a triple store

nodes. An RDF blank node is an RDF node, which itself does not contain any data, but serves as a parent node to a grouping of data. From an RDF/XML syntactical standpoint, a blank node is an *rdf:Description* element that does not have an *rdf:about* attribute assigned to it. A blank node can perform two tasks in an RDF graph; it can be the object in one RDF statement and the subject in another. However due to their empty description (it has no IRI), blank nodes are anonymously extracted in the view when queried. Intermediate results with blank nodes are then not possible for query languages except for vSPARQL. Also, the fact that ontology query languages are all based on SPARQL, which was designed for RDF queries, is a brake to easily extract sub-ontologies from $\mathcal{SHOIN}(\mathcal{D})$ or OWL DL ontologies. Indeed the syntax can be complicated to adapt when selecting specific $\mathcal{SHOIN}(\mathcal{D})$ axioms. More generally, the definition of a view with a query is much more difficult than the definition of a mapping with a subset extraction approach, as it is a declarative approach. Unlike query approaches, the advantage of subset extraction, (i.e. manual, structural and logical approaches) on the ontology is that it be done simply and visually. Besides the precision of the extraction is more or less high depending on the tools provided by the extraction approach. For example, a traversal specification will allow specifying a starter concept in the ontology and a depth for the traversal function, so the precision will be limited to this depth. In the case of manual subset extraction based on the aggregation of one by one element extractions, the precision level will always be maximal. Regarding module extraction techniques, they may produce modules containing data that the user is not interested in. This could be because a non-minimal module was produced. A minimal module is a module that provides the same description of the relationships between terms over a given sub-vocabulary as the whole ontology. However, if a non-minimal module is produced, the user must inspect the extracted subset to determine the concepts to forget. Unlike these subset extraction approaches, precision level is not a problem with a query-based approaches as the query allows to precisely select the elements wanted in the ontology.

Second, the **manageable quality of the portion extracted** depends a lot considering the two main types of approaches. By manageable, we mean that the sub-graph can be easily accessed and queried (like the ontology), updated (like database views) and evolved (like ontologies). On one hand, updating a view specified by a query, in response to changes in the ontology, just implies updating the query. On the other hand, updating a view specified by an extraction approach implies redefining the manual extraction since the beginning and then reapplying the extraction operation on the updated ontology. Extraction approaches require significant user effort and must be repeated whenever the information set is updated. In the case of module extractions, if some users want to modify or transform the data that they extract from an ontology, the approach does not allow to specify changes to be made to the ontology before or during extraction; you need to modify the original data and then derive a module from the materialized modified ontology. These approaches are purely extraction techniques; any modification or restructuring of the information must be performed externally on a materialized copy of the output. This modification issue is handled by ontology query approaches by simply editing the query. To sum up, query based approaches allow the extraction of a sub-graph, which is manageable by update of its definition (query) but is not manageable on its own.

Third, the question of the **nature of the portion of ontology extracted** is still open. Does the view have to extract only sub-graphs or sub-ontologies? Logical and module extraction approaches, because they are based on description logics (DL), guaranty the extraction of a DL sub-ontology on which knowledge can be inferred, which is not always the case in query based approaches. Indeed query approaches are not specifically designed for deriving subsets of the original ontology that have the same meaning as the original one for a specified signature. Some users may want to simply extract lists of terms or small connected sub-graphs and may not need to extract ontologies. However in the literature, more and more researchers tend to think that a **view taken from the ontology is not just a portion of the ontology, but also a collection of concepts and relationships, and is itself a new interpretation of the ontology** (Rajugan & al, 2006). This means that this portion of ontology is itself an ontology different from its source, specialized for new perspectives of use, evolving in a new context.

The questions of an optimal extraction of portion of an ontology, its management and its nature should be dealt before any agreement is found on the definition of a view. As we can see issues concerning query based approaches can be resolved by subset extraction approaches, and inversely. An idea could then be to merge the two techniques in a hybrid approach. For instance depending on the level of simplification wanted for the definition of the view, the user could choose to directly compose a query (if he wants a simple sub-graph) or use a graphical subset extractor or module extractor (if he wants a sub-ontology) on the ontology, which would transcript the extraction mapping into a query that could be stored. The result would however depend on the expressivity of the query language chosen. In this case, the definition of the view could be edited and updated directly with the query if required. The sub-graph, if corresponding to a sub-ontology, could nevertheless be updated, evolved and enriched if the user wants to consider it as an independent ontology and apply it to a specific application domain. In this case, the link between the source ontology and the evolved sub-ontology would be broken but not the link between the source and the previous one. However, the question of the view dependence to the ontology has to be studied with respect to the use perspectives of each project.

To fully cope with the Ontology Change Management requirement, a **view should ideally be extracted, managed and evolved as an ontology** to allow parallel or branch evolutions of ontologies that can be managed at the same time, reinforcing ontology versioning features. Logical approaches seem to well fit this requirement. The contribution of these methods to the goal of my thesis lies in their ability to **adapt an ontology to one or more user assumptions and extract a precise and consistent specialized portion of the ontology** in a much easier way than ontology evolution operators. Logical view extraction approaches can produce **sub-ontologies from the original ontology specified in the same formal language.** However, **existing change management methodologies are not feasible with these view extraction approaches**, because these approaches do not use and represent ontological changes used for the view extraction. Versioning of the extraction of such sub-ontologies is then limited to version identifiers as no log of change is provided. Propagation of changes between the ontology and the different sub-ontologies, required for the 7th criterion, is not addressed either. In order to cope with my objectives, my proposal must be able to **convert the logical extraction and management of ontology views into ontological changes traceable once the extraction process is completed**.

3 Conclusion: Deadlocks and Approach of my Proposal

This third section first recalls the eight criteria identified to fulfil the three objectives of my thesis and the corresponding solutions inspired by the existing works in OCM and Ontology Views' research fields. It deduces the remaining deadlocks my proposal must solve. Secondly, it presents the approach chosen to reach the objectives.

Previous sections have identified many of the key solutions that would allow to satisfy the eight criteria in my proposal:

1. A formal ontology evolution methodology enabling conceptual modelling following an ontological analysis, can be derived from Klein and Stojanovic's proposals.
2. Specifying ontology and changes in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal language, can be achieved by deriving ontological changes from the $\mathcal{SHOIN}(\mathcal{D})$ language model like Klein and Stojanovic's proposals.
3. Managing a formal ontology representing the several user assumptions on the domain can be realized with the specifications of ontology views using a logical approach inspired by (Kontchatkov & al, 2009, Grau & al, 2007, Grau & al, 2008).
4. Managing the logical consistency in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages can be guided by a preventive logical consistency resolution approach, based on the belief contraction of the evolution log inspired by (Flouris, 2006). It has to be verified by a logical consistency corrective approach with the use of a $\mathcal{SHOIN}(\mathcal{D})$ dedicated reasoner like Pellet.
5. Management of structural consistency in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages with formal representation of changes, can be first guarantied by the use of a fixed $\mathcal{SHOIN}(\mathcal{D})$ structure for which the list of allowed changes is limited. Second it is ensured by the verification of structural constraints derived from the $\mathcal{SHOIN}(\mathcal{D})$ axiom dependencies after the representation of the changes to apply like (Pittet & al, 2013).
6. Evolution and Versioning of a formal ontology and its user assumptions, can be handled with the logical specification of ontology views, producing sub-ontologies that can also evolve and be versioned like ontologies according to Klein and Stojanovic's proposals.
7. Impact analysis & change propagation from the evolved ontology to the user assumptions, and inversely, can be handled if the views extractions are traced in the ontology log of changes and if the views also have their own log of changes. This problem has however not been addressed in existing works.
8. Adaptation of the ontology to a new user assumption can be realized with the specification of a new user assumption on the ontology with a user-friendly specification of a view inspired by (Noy & Musen, 2004) traversal views.

Considering the 7th criterion that still remains unsatisfied in existing works, the main challenge for my proposal is to design a methodology that **extends OCM features to management of ontology views and change propagation between them and their source ontology**. My proposal should then address the following three deadlocks:

1. Extraction of $\mathcal{SHOIN}(\mathcal{D})$ ontology views modelled as an ontological change, traceable in the ontology log of changes
2. The creation of a log of changes for each ontology views, to allow them to evolve and be versioned like the original $\mathcal{SHOIN}(\mathcal{D})$ ontology.
3. The impact management by the propagation of the changes applied from the ontology to its views and from a view to the ontology and the other views, based on the log of changes of each of them.

To cope with these three objectives, the approach is articulated in three blocks.

The first block consists in the definition of a **collaborative change management process ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology**. It should follow the six phases of the evolution process defined by (Stojanovic, 2004). An ontology model suited for $\mathcal{SHOIN}(\mathcal{D})$ should be formalized to model structurally consistent changes by guarantying the expressivity level of the ontology after their application. An additional structural consistency preventive approach based on axiom dependencies resolution should be performed at the end of the change modelling phase. A preventive logical consistency qualification of these changes should be included at the change semantics phase in order to help the user ensure their logically consistent application. A log of changes should be generated and updated after each validated phase in order to display the versioning features of the methodology such as change propagation. **This first block is described in Chapter 4 and Chapter 5**. Chapter 4 formalizes the $\mathcal{SHOIN}(\mathcal{D})$ ontology model and its related basic and composed changes modelling. Chapter 5 formalizes the preventive structural an logical consistency approaches

The second block consists in the **logical specification and extraction of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. The definition of a view should be traceable like a change in the ontology global log of changes. A global log of changes should be generated for each view extracted in order to display the versioning features of the methodology on the views. **This second block is described in Chapter 6**.

The third block consists in the **impact management with the consistent propagation of changes** applied from the ontology to its views (Top-Down approach) and, inversely from a view to the ontology and the other views (Bottom-Up Approach). **This third block is described in Chapter 7**.

References

Alkhateeb, F., Baget, J. F., & Euzenat, J. (2009). Extending SPARQL with regular expression patterns (for querying RDF). Web Semantics: Science, Services and Agents on the World Wide Web, 7(2), 57-73.

Allocca, C., d'Aquin, M., & Motta, E. (2009). Detecting different versions of ontologies in large ontology repositories.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., ... & Sherlock, G. (2000). Gene Ontology: tool for the unification of biology. Nature genetics, 25(1), 25-29.

Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., ... & Zacharias, V. (2002). KAON—towards a large scale Semantic Web. In E-Commerce and Web Technologies (pp. 304-313). Springer Berlin Heidelberg.

Ceravolo, P., Corallo, A., Damiani, E., Elia, G., Viviani, M., & Zilli, A. (2006). Bottom-up extraction and maintenance of ontology-based metadata. Capturing Intelligence, 1, 265-282.

Choi, N., Il-Yeol, S. & Han, H. 2006. "A Survey on Ontology Mapping" ACM SIGMOD Record, 35(3), pp. 34-41.

De Bruijn, J., Martin-Recuerda, F., Manov, D. & Ehrig, M. 2004. "D4.2.1: State of the Art Survey on Ontology.

Merging and Aligning" available on the Web: http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/debruijn04state.pdf

Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuiness, D., Patel-Schneider, P. & Stein, L. A. 2004. "OWL Web Ontology Language Reference" W3C Recommendation, available on the Web at http://www.w3.org/TR/owl-ref/

Decker, S., & Frank, M. (2004, May). The social semantic desktop. In WWW2004 Workshop Application Design, Development and Implementation Issues in the Semantic Web (Vol. 9, p. 10).

Degtyarenko, K., De Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., ... & Ashburner, M. (2008). ChEBI: a database and ontology for chemical entities of biological interest. Nucleic acids research, 36(suppl 1), D344-D350.

Detwiler, L.T. Shaw, M. and Brinkley, J.F., (2010) Ontology View Query Management. In Proceedings, Annual Symposium of the American Medical Informatics Association, pages 1023, Washington, DC.R.

Djedidi, R. (2009). Approche d'évolution d'ontologie guidée par des patrons de gestion de changement (Doctoral dissertation, Université Paris Sud-Paris XI).

Djedidi, R., & Aufaure, M. A. (2010). ONTO-EVO A L an ontology evolution approach guided by pattern modeling and quality evaluation. In Foundations of Information and Knowledge Systems (pp. 286-305). Springer Berlin Heidelberg.

Do, H. H., & Rahm, E. (2004). Flexible integration of molecular-biological annotation data: The genmapper approach. In Advances in Database Technology-EDBT 2004 (pp. 811-822). Springer Berlin Heidelberg.

Eder, J., & Koncilia, C. (2004, January). Modelling changes in ontologies. In On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops (pp. 662-673). Springer Berlin Heidelberg.

Euzenat, J., Le Bach, T., Barrasa, J., Bouquet, P., de Bo, J., Dieng, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckeschmidt, H., Shvaiko, P., Tessaris, S., van Acker, S. & Zaihrayeu, I. 2004. "D2.2.3: State of the Art on Ontology Alignment" available on the Web (last visited November, 2007): http://www.starlab.vub.ac.be/research/projects/knowledgeweb/kweb-223.pdf

Euzenat, J. & Shvaiko, P. 2007. Ontology Matching, Springer-Verlag.

Flouris, G. (2006). On belief change in ontology evolution: Thesis. AI Communications, 19(4), 395-397.

Flouris, G., Plexousakis, D., & Antoniou, G. (2005). On applying the AGM theory to DLs and OWL. In The Semantic Web–ISWC 2005 (pp. 216-231). Springer Berlin Heidelberg.

Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G. (2008). Ontology change: Classification and survey. The Knowledge Engineering Review, 23(02), 117-152.

Grandi, F. (2009). Multi-temporal RDF ontology versioning. In Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD-09).

Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2007, May). Just the right amount: extracting modules from ontologies. In Proceedings of the 16th international conference on World Wide Web (pp. 717-726). ACM.

Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2008). Modular Reuse of Ontologies: Theory and Practice. J. Artif. Intell. Res.(JAIR), 31, 273-318.

Gueffaz, M., Pittet, P., Rampacek, S., Cruz, C., & Nicolle, C. (2012). Inconsistency Identification In Dynamic Ontologies Based On Model Checking. INSTICC, ACM SIGMIS, 418-421.

Haase, P., & Stojanovic, L. (2005). Consistent evolution of OWL ontologies. In The Semantic Web: Research and Applications (pp. 182-197). Springer Berlin Heidelberg.

Haase, P., Van Harmelen, F., Huang, Z., Stuckenschmidt, H., & Sure, Y. (2005). A framework for handling inconsistency in changing ontologies. In The Semantic Web–ISWC 2005 (pp. 353-367). Springer Berlin Heidelberg.

Hadzic, M., & Chang, E. (2004). Role of the ontologies in the context of grid computing and application for the human disease studies. In Semantics of a Networked World. Semantics for Grid Databases (pp. 316-318). Springer Berlin Heidelberg.

Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In The Semantic Web-ISWC 2003 (pp. 17-29). Springer Berlin Heidelberg.

Jaziri, W. (2009, October). A methodology for ontology evolution and versioning. In Advances in Semantic Processing, 2009. SEMAPRO'09. Third International Conference on (pp. 15-21). IEEE.

Kalfoglou, Y. & Schorlemmer, M. 2003. "Ontology Mapping: the State of the Art" Knowledge Engineering Review,18 (1), pp. 1-31.

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., & Scholl, M. (2002, May). RQL: a declarative query language for RDF. In Proceedings of the 11th international conference on World Wide Web (pp. 592-603). ACM.

Klein, M. C. A. (2004). Change management for distributed ontologies.

Klein, M. & Fensel, D. 2001. "Ontology Versioning on the Semantic Web" Proceedings of the International Semantic Web Working Symposium (SWWS), pp. 75-91.

Klein, M., Fensel, D., Kiryakov, A., & Ognyanov, D. (2002). Ontology versioning and change detection on the web. In Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web (pp. 197-212). Springer Berlin Heidelberg.

Klein, M., & Noy, N. F. (2003, March). A component-based framework for ontology evolution. In Proceedings of the IJCAI (Vol. 3).

Kochut, K. J., & Janik, M. (2007). SPARQLeR: Extended SPARQL for semantic association discovery. In The Semantic Web: Research and Applications (pp. 145-159). Springer Berlin Heidelberg.

Kontchakov, R., Pulina, L., Sattler, U., Schneider, T., Selmer, P., Wolter, F., & Zakharyaschev, M. (2009, July). Minimal Module Extraction from DL-Lite Ontologies Using QBF Solvers. In IJCAI (Vol. 9, pp. 836-841).

Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., & Morbidoni, C. (2009, April). Rapid prototyping of semantic mash-ups through semantic web pipes. In Proceedings of the 18th international conference on World wide web (pp. 581-590). ACM.

Liang, M. (2006). Enabling active ontology change management within semantic web-based applications.

Luong, P. H. (2007). Gestion de l'évolution d'un Web sémantique d'entreprise (Doctoral dissertation, École Nationale Supérieure des Mines de Paris).

Magkanaraki, A., Tannen, V., Christophides, V. and Plexousakis. D. Viewing the semantic web through RVL lenses. In Second International Semantic Web Conference, volume 2870, pages 96–112, Sanibel Island, FL, 2003. Springer-Verlag.

Maynard, D., Peters, W., d'Aquin, M., & Sabou, M. (2007, June). Change management for metadata evolution. In International Workshop on Ontology Dynamics (IWOD-07) (p. 27).

Miklós, Z., Neumann, G., Zdun, U., & Sintek, M. (2003). Querying semantic web resources using triple views (pp. 517-532). Springer Berlin Heidelberg.

Noy, N. F., & Klein, M. (2004). Ontology evolution: Not the same as schema evolution. Knowledge and information systems, 6(4), 428-440.

Noy, N. F., & Musen, M. A. (2002). Promptdiff: A fixed-point algorithm for comparing ontology versions. AAAI/IAAI, 2002, 744-750.

Noy, N. F., & Musen, M. A. (2004). Specifying ontology views by traversal. In The Semantic Web–ISWC 2004 (pp. 713-725). Springer Berlin Heidelberg.

Obrst, L., Liu, H., & Wray, R. (2003). Ontologies for corporate web applications. AI Magazine, 24(3), 49.

Pérez, J., Arenas, M., & Gutierrez, C. (2010). nSPARQL: A navigational language for RDF. Web Semantics: Science, Services and Agents on the World Wide Web, 8(4), 255-270.

Pinto, H.S., Gomez-Perez, A. & Martins, J. P. 1999. "Some Issues on Ontology Integration" Proceedings of the Workshop on Ontologies and Problem-Solving Methods (KRR5) at 16 th International Joint Conference on Artificial Intelligence (IJCAI-99).

Plessers, P. (2006). An Approach to Web-based Ontology Evolution.

Polleres, A., Scharffe, F., & Schindlauer, R. (2007). SPARQL++ for mapping between RDF vocabularies. In On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS (pp. 878-896). Springer Berlin Heidelberg.

Rajugan, R., Chang, E., & Dillon, T. S. (2006, January). Ontology views: A theoretical perspective. In On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops (pp. 1814-1824). Springer Berlin Heidelberg.

Ribeiro, M. M., & Wassermann, R. (2009a). AGM revision in description logics. Proceedings of ARCOE.

Ribeiro, M. M., & Wassermann, R. (2009b). Base revision for ontology debugging. Journal of Logic and Computation, 19(5), 721-743.

Ribeiro, M., Wassermann, R., Antoniou, G., Flouris, G., & Pan, J. (2009, September). Belief contraction in web-ontology languages. In Proceedings of the 3rd International Workshop on Ontology Dynamics, IWOD.

Rogozan, C. D. (2009). Gestion de l'évolution des ontologies: méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions d'ontologie (Doctoral dissertation, Université du Québec à Montréal).

Rosse, C., & Mejino Jr, J. L. (2003). A reference ontology for biomedical informatics: the Foundational Model of Anatomy. Journal of biomedical informatics, 36(6), 478-500.

Schenk, S., & Staab, S. (2008, April). Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In Proceedings of the 17th international conference on World Wide Web (pp. 585-594). ACM.

Seidenberg, J., & Rector, A. (2006, May). Web ontology segmentation: analysis, classification and use. In Proceedings of the 15th international conference on World Wide Web (pp. 13-22). ACM.

Schlobach, S. & Cornet, R. 2003. "Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies" Proceedings of the 18 th International Joint Conference on Artificial Intelligence (IJCAI-03)

Shaw, M., Detwiler, L. T., Noy, N., Brinkley, J., & Suciu, D. (2011). vSPARQL: A view definition language for the semantic web. Journal of biomedical informatics, 44(1), 102-117.

Sintek, M., Van Elst, L., Scerri, S., & Handschuh, S. (2007). Distributed knowledge representation on the social semantic desktop: Named graphs, views and roles in nrl. In The Semantic Web: Research and Applications (pp. 594-608). Springer Berlin Heidelberg.

Stojanovic, L. (2004). Methods and tools for ontology evolution.

Stojanovic, L., Maedche, A., Motik, B. & Stojanovic, N. 2002. "User-driven Ontology Evolution Management" Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW-02), Lecture Notes in Computer Science (LNCS), Volume 2473, Springer-Verlag, pp. 285-300.

Volz, R., Oberle, D., & Studer, R. (2003, March). Views for light-weight web ontologies. In Proceedings of the 2003 ACM symposium on Applied computing (pp. 1168-1173). ACM.

Westerski, A. (2009). Integrated environment for visual data-level mashup development. In Web Information Systems Engineering-WISE 2009 (pp. 481-487). Springer Berlin Heidelberg.

Wouters, C. "A Formalization and Application of Ontology Extraction," School ofEngineering and Mathematical Sciences Faculty of Sciences, Technology and Engineering, La Trobe University, Melbourne, Australia, Melbourne, Doctor of Philosophy (Ph.D) thesis, 2006. pp. 460.

Yildiz, B. (2006). Ontology Evolution and Versioning The state of the art.

Chapter 4

$\mathcal{SHOIN}(\mathcal{D})$ Collaborative OCM: Methodology and Model

Summary

This chapter aims to introduce and formalize the methodology and model designed for the **block (1)** of my proposal, i.e. the definition of a **collaborative OCM methodology ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology**. A first section describes the **global OCM methodology, called OntoVersionGraph,** through the different phases of its evolution process. A second section formalizes the **$\mathcal{SHOIN}(\mathcal{D})$ Ontology Model** suited for change management on which my whole proposal is based. A third section formalizes the **change modelling** task of the change management process according to the proposed model. Basic and composed changes are defined as operations adding and deleting axioms from the ontology structure according to the model. The definitions of ontology log of changes and global log of changes, in which changes are represented and traced to ensure OCM features, are formalized.

Plan

1 OntoVersionGraph: A $\mathcal{SHOIN}(\mathcal{D})$ Ontology Collaborative Change Management Methodology 91
1.1 Change Detection Phase 91
1.2 Change Modelling Phase 93
1.3 Change Semantics Phase 93
1.4 Change Implementation Phase 94
1.5 Change Propagation Phase 94
1.6 Change Validation Phase 95
1.7 Discussion 96
2 SHOIN(D) Ontology Model 97
2.1 Preliminary Notions 97
2.2 Model definition of the structure for a SHOIN(D) Ontology 98
2.3 SHOIN(D) Ontology Example 99
3 SHOIN(D) Ontology Change Modelling 104
3.1 Change Definitions 104
3.2 SHOIN(D) Basic Changes Representation 105
3.3 SHOIN(D) Composed Changes Representation 106
4 Conclusion 108

Previous chapter has identified the three blocks composing the approach of my thesis. This chapter focuses on the methodology and model designed for the first block of my proposal, i.e. the definition of a **collaborative change management process ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ formal ontology**. In this thesis, we call **$\mathcal{SHOIN}(\mathcal{D})$ ontology** any ontology specified in a language whose expressivity level is equivalent to $\mathcal{SHOIN}(\mathcal{D})$ description logic. Following the Klein's change management definition (Klein, 2004), it combines ontology evolution and versioning activities. The **evolution process** is divided in the six phases recommended by Stojanovic in (Stojanovic, 2004) and its validation is conditioned by a **collaborative agreement from domain experts**. The methodology can be applied to build and evolve any $\mathcal{SHOIN}(\mathcal{D})$ ontology from scratch. The construction of a global change log tracing every change implemented since its creation allows performing **versioning** and **consistency maintenance tasks**. Existing ontologies can also be evolved. A reconstruction phase of the ontology global log of changes is required to display them. To support the specification of multiple assumptions for the domain users and provide them access to the corresponding sub-ontologies, all along the ontology lifecycle, the methodology is improved with **ontology views specification** and **their corresponding change impact management**. Views, providing such sub-ontologies, allow decreasing the ontology complexity by cutting off the ontology knowledge, which is not reused by domain users.

1 OntoVersionGraph: A $\mathcal{SHOIN}(\mathcal{D})$ Ontology Collaborative Change Management Methodology

This first section describes the methodology, called OntoVersionGraph, through its six evolution process phases as illustrated in Figure 14: **change detection, change modelling, change semantics, change propagation, change implementation,** and **change validation**.

1.1 Change Detection Phase

According to (Stojanovic, 2004), the **change detection phase** consists in **evaluating the need of change** for the ontology evolution. As seen in Chapter 3, evolution changes can be caused by a need of change in the domain or in the conceptualization. Considering a $\mathcal{SHOIN}(\mathcal{D})$ ontology, the changes can impact the terminological and the assertional level, i.e. the **whole knowledge base**. As seen in Chapter 2, formal ontology development and, by extension, ontology evolution, require an **ontological analysis** before any **conceptual modelling** is realized. Ontological analysis, which analyses each assumption supported by a user of the domain, can therefore be part of the change detection phase. If the result of the ontological analysis shows changes in the way the domain is seen by its community, then there are **conceptual changes** to take into account. The domain experts must start and agree on a new conceptual modelling to update the ontology conceptualization and consequently its terminology. If changes occurred directly in the domain content itself, without questioning the ontology conceptualization and the domain user assumptions, then it corresponds to **assertional changes**. These types of changes do not really need a long discussion between experts, i.e. no conceptual modelling, they only imply adding or deleting assertions to the knowledge base according to new facts occurring in the domain.

From these observations, the **change detection phase cannot be automated** if embedding an ontological analysis activity. As such, it is the unique phase, which is not semi-automated in the OntoVersionGraph evolution process (c.f. Fig 14 Change Detection). The change detection phase process is composed of the following steps. First, an ontological analysis is done on the domain. Second, its results allow experts of the domain establishing, in a consensus, an **informal list of conceptual and assertional changes** required for the ontology.

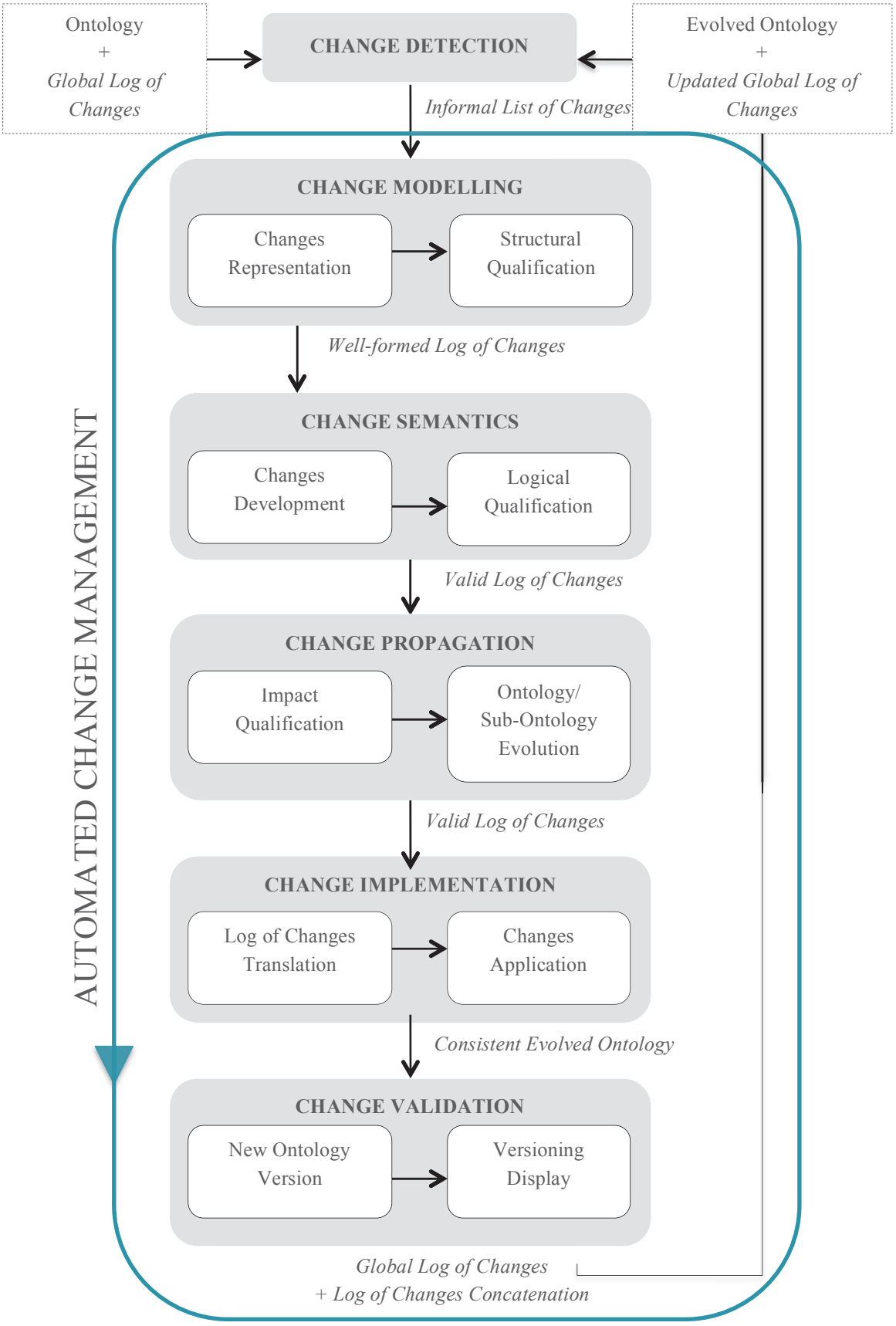


Figure 14: OntoVersionGraph Evolution Process

1.2 Change Modelling Phase

The **change modelling phase**, called change representation phase in (Stojanovic, 2004), aims at **modelling the changes in a suitable format**. First, it must operate the translation of the informal detected changes into **formal ontological operations**. Ontological operations correspond to **additions or deletions of terminological and assertional axioms** in the ontology. According to (Klein, 2004) and (Stojanovic, 2004), to make sure these operations respect the language model, each operation type has to be tailored from the considered language allowed constructs. Also, for convenience, **composed operation types** considered as usual operations composed of **fine grained operation types,** called **basic operation types**, can be modelled and applied as one operation. Second, the modelled basic and composed operations have to be checked for **structural consistency**, in order to guaranty their compliance with the ontology model. According to (Stojanovic, 2004), a structural consistency preventive approach can be used here to resolve any structural issue. Third the structural consistent operations have to be represented in order to be later traced by the **incremental versioning system**.

On this basis, the OntoVersionGraph methodology provides a $\mathcal{SHOIN}(\mathcal{D})$ **ontology model** suited for change modelling presented in the second section of this chapter. It provides a list of **addition and deletion basic change** operator types limited by the $\mathcal{SHOIN}(\mathcal{D})$ allowed constructs. Composed change types can be built by aggregating several basic change types or other composed change types. OntoVersionGraph also embeds a $\mathcal{SHOIN}(\mathcal{D})$ **structural consistency preventive approach** to resolve remaining structural issues of the modelled changes and provide a structurally consistent list of changes. As said in Chapter 3, the solution chosen to represent and trace changes is the **evolution log**. Two types of evolution logs are defined. A **log of changes** is used to represent changes to apply on the ontology during the evolution process. A global log, concatenating each log of changes issued from the whole ontology evolution, called the **ontology global log of changes**, traces changes applied on an ontology since its creation. The change modelling process (c.f. Figure 14 Change Modelling) comes along the following steps. First, ontology domain experts match the changes of the informal list issued from the change detection phase with the list of $\mathcal{SHOIN}(\mathcal{D})$ basic changes. If required, composed changes can be modelled by aggregating basic ones. Second, their structural consistency is checked and resolved according to the $\mathcal{SHOIN}(\mathcal{D})$ structural consistency preventive approach. Fourth, the **list of structurally consistent changes** is represented as such in a log of changes.

1.3 Change Semantics Phase

According to (Stojanovic, 2004), the **change semantics phase** aims at **helping the user predict the effects of the changes** that have to be applied on the ontology, namely in terms of **logical consistency**. Stojanovic also advises the use of a **richer description of the changes** in order to determine semantic role of ontology entities. Composed changes should then be decomposed into their composing basic ones to allow visualizing them at a finer grained level and turn them semantically understandable. She also recommends the use of a **logical consistency preventive approach** in order to guide the user for the identification and resolution of logical inconsistencies the application of changes could potentially cause. If inconsistencies are detected, the user will have to revise the change modelling by undertaking additional change detection and change modelling phases. If the inconsistencies concern assertional changes, i.e. changes adding an assertion, the resolution often can be done without revising the ontology conceptualization. If they concern conceptual changes, then conceptualization has to be revised. Considering a formal ontology, the resolution task has to be **collaboratively realized by the domain experts** until a new agreement is found. When logical inconsistency is resolved according to the preventive approach, changes are potentially assumed **consistent**.

From these findings, the change semantics phase proposed by the OntoVersionGraph methodology embeds a logical consistency preventive approach to help domain experts identifying and

resolving potential logical inconsistencies. This semi-automated approach, described in Chapter 5, is inspired by the **belief contraction approach** described in Chapter 3 and uses a **fixed set of logical unsatisfiability constraints** for each of the $\mathcal{SHOIN}(\mathcal{D})$ allowed basic change types. Composed changes represented in the log of changes are decomposed into basic ones, in order to build their unsatisfiability constraints from the $\mathcal{SHOIN}(\mathcal{D})$ basic change types ones. If a basic change is identified as not compliant according to the whole set of unsatisfiability constraints from the ontology global change log, then the **minimal inconsistent set of changes** in cause is returned to the domain experts. These experts can then discuss of the resolution and, if required, undertake a new change detection phase by revising the conceptual modelling, followed by a new change modelling phase. When changes are assumed consistent according to the unsatisfiability constraints of the whole ontology global log of changes, the **log of changes is considered logically consistent** and can be reused for the propagation phase. The change semantics phase process (c.f. Fig.14 Change Semantics) comes along the following steps. First, changes are developed into basic changes and their unsatisfiability constraints evaluated. Second, these unsatisfiability constraints are unified with the unsatisfiability constraints of the previous logs of changes concatenated in the ontology global log of changes. They are all instantiated with elements names of the ontology in order to generate the potential inconsistent sets of changes. Third if any potential set of changes is found in the developed log of changes, then it is returned to the domain experts.

1.4 Change Implementation Phase

According to (Stojanovic, 2004), the **change implementation phase** consists in **applying the verified changes to the ontology**, such that the terminological and the assertional levels of the knowledge base are updated. In (Stojanovic, 2004), this phase follows a propagation phase, which is used to propagate conceptual changes from the terminology to the instances. Considering a change semantics phase in OntoVersionGraph, which ensures logical consistency on both terminological and assertional levels, propagation to instances is already resolved. Yet, considering a formal ontology with ontology views, a change propagation phase is required to resolve the change impact between them. But **changes have to be firstly implemented on the ontology before propagated to its views**, as the corresponding sub-ontologies have to be extracted from the updated ontology.

According to these guidelines, OntoVersionGraph embeds a change implementation phase, which simply **applies changes on the ontology terminological and assertional axioms**. However these are not directly **applied to** the original ontology but to **a copy** of it. Until ontology evolution validation, the ontology and its views should indeed remain the same to stay accessible for domain users. If changes are implemented on the original version, before change propagation to the ontology views, the domain users querying the views will not be able to access the updated knowledge of the ontology. The change implementation phase (c.f. Fig.14 Change Implementation) follows two steps. First it generates a copy of the ontology and all its views. Second it applies the changes on the duplicated ontology. The evolved copy of the ontology is then ready to be checked by a **logical consistency corrective approach** to detect the logical inconsistencies that potentially may have been forgotten by the preventive approach. The OntoVersionGraph methodology uses the **Pellet reasoner** to achieve this task. If logical inconsistencies are detected by the reasoner, then a new change modelling phase is launched. The change propagation phase cannot be started until the ontology is not assumed as structurally and logically consistent.

1.5 Change Propagation Phase

According to (Djedidi, 2008), **change propagation** should **evaluate and resolve the impact of changes to any artefact depending on the ontology**. Inspired by (Stojanovic, 2004) change propagation of conceptual changes to the knowledge base assertional level, the change propagation phase proposed in many OCM methodologies focuses on dependant ontologies impact resolution. In OntoVersionGraph,

considering ontology views extracting sub-ontologies, which are true sub-portions of a formal ontology, the change propagation phase can therefore focus on their impact. **Resolving the impact of changes on such ontology views** first amounts to consistently identify the right set of changes to apply among those implemented on their source ontology. View sub-ontologies, being themselves ontologies, can also evolve. Therefore, resolving the impact of changes secondly **implies propagating their own changes to the source ontology**.

On this basis, the OntoVersionGraph methodology proposes a **change propagation phase** (c.f. Fig. 14 Change Propagation) **dedicated to $\mathcal{SHOIN}(\mathcal{D})$ ontology views**. $\mathcal{SHOIN}(\mathcal{D})$ ontology views specified on an ontology, being themselves ontologies with a global log of changes, can be easily updated by propagating changes of the log of changes of the evolving ontology (c.f. Top Down Approach in Chapter 6). Inversely, their evolution provides a log of changes, which changes can be propagated to their source ontology (c.f. Bottom-Up Approach in Chapter 6). It is important to note that in this case, the ontology will evolve following the evolution process and another change propagation phase will resolve the impact with the other views.

The **change propagation phase process from an ontology to its views,** called **Top-Down Propagation approach** comes along the following steps. First, the **impact of the log of changes,** issued from the change semantics phase of the ontology evolution, is assessed according to the view definition. Second, if impacted, **the definition of the view is revised**. Third the corresponding sub-ontology is updated by a **new extraction on the evolved ontology**, according to its updated definition. Until the change validation phase each new extracted sub-ontology remains as a copy of the current ones.

The **change propagation phase process from a view to its source ontology,** called **Bottom-Up Propagation approach,** follows these steps. First, the log of changes issued from the view sub-ontology evolution is reused to start a **new source ontology evolution**. Second, the structural and logical consistency preventive approaches of the respective change modelling and change semantics phases assess the changes. If structural consistencies are found, they are automatically resolved. But if logical inconsistencies are detected, the **source ontology and the previous view evolution processes stop**. The **minimal sets of inconsistent changes are returned to the domain experts of the views impacted** by these inconsistencies. They have to collaboratively agree on a new modelling of the changes before restarting new evolution processes for the view and the source ontology. Third when the new modelled changes are assumed consistent on both view and source ontology, the **source ontology change propagation phase spreads and manage the impact the changes to the other views** using the Top Down approach.

1.6 Change Validation Phase

According to (Stojanovic, 2004), the **change validation phase** consists in **committing the implemented changes** such that the ontology is replaced by its evolved version and the knowledge base is updated. To display the incremental versioning features, the **previous ontology version or a record of the changes applied has to be stored**. This may allow to compare versions of the ontology, retrieving or rebuilding a previous version, roll backing changes etc.

From these observations, OntoVersionGraph provides a change validation phase (c.f. Fig. 14 Change Validation), which allows the domain experts to **validate all the current evolution processes,** which have not been stopped**.** The corresponding evolved copy ontology and its updated views are committed as **new versions**. The new ontology and/or views versions replace the current ones. The **current global logs of changes are archived** as previous versions in the versioning system. The ontology and/or views logs of changes issued from the validated evolution processes are concatenated to copies of the corresponding archived global logs of changes, producing **new global logs of changes**. The new ontology and/or views global logs of changes are stored in the versioning system.

1.7 Discussion

This section has described the $\mathcal{SHOIN}(\mathcal{D})$ collaborative change management methodology, called OntoVersionGraph, through six phases as recommended by (Stojanovic, 2004). However, as the propagation phase of our methodology does not handle the propagation of conceptual changes on the instances – it is already considered during the change modelling phase - but aims at propagating the changes to the dependent ontologies or sub-ontologies, it appears after the change implementation phase. The second logical consistency check has to confirm that the changes implemented on the copy of the ontology preserve its consistency before any propagation is envisioned. This is the main contribution of our evolution process considering the (Stojanovic, 2004)'s one. This section has also allowed to locate in the evolution process the modules of the three blocks of our methodology, which require a specific formalization. The **$\mathcal{SHOIN}(\mathcal{D})$ ontology model** is required for the change modelling phase. The **$\mathcal{SHOIN}(\mathcal{D})$ structural consistency preventive approach** is also needed by the change modelling phase. The **$\mathcal{SHOIN}(\mathcal{D})$ logical consistency preventive approach** is necessary for the change semantics phase. The **change impact management** between $\mathcal{SHOIN}(\mathcal{D})$ ontology and its views is mandatory for the change propagation phase. The **$\mathcal{SHOIN}(\mathcal{D})$ ontology view extraction approach** is the only module, which does not appear in the evolution process. View extraction is indeed not an evolution task as such, as it does not apply a change on an ontology but create a sub-ontology from it. However it is a crucial module to formalize for the OntoVersionGraph methodology as the other modules used in the evolution process depend on the extraction and representation of consistent $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies.

2 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model

This section presents the **$\mathcal{SHOIN}(\mathcal{D})$ ontology model** required in the change modelling phase of the OntoVersionGraph evolution process. It is important to note that this model is the basis of the formalization of the whole blocks of my proposal. The first part gives the preliminary notions for the definition of the $\mathcal{SHOIN}(\mathcal{D})$ ontology model. A second part formalizes the $\mathcal{SHOIN}(\mathcal{D})$ ontology model as a mathematical fixed structure built upon the constructs allowed by the $\mathcal{SHOIN}(\mathcal{D})$ language. A third part illustrates an application of the model on an example $\mathcal{SHOIN}(\mathcal{D})$ ontology inspired by the UOBM Ontology Benchmark (Ma & al, 2006).

2.1 Preliminary Notions

This part introduces preliminary notions for the definition of the $\mathcal{SHOIN}(\mathcal{D})$ ontology model proposed in this section. The first paragraph deals with the $\mathcal{SHOIN}(\mathcal{D})$ description logic specificities to take into account for the representation of an ontology. The second paragraph gives the mathematical basis used to formalize the model.

According to (Baader, 2003), **description logics** (DLs) are a family of logics that are decidable fragments of first-order logic (FOL) with attractive and well-understood computational properties. DLs have been in use for over two decades to formalize knowledge and notably quality ontologies. In a DL knowledge base, a distinction is drawn between the so-called **T-Box** (terminological level) and the **A-Box** (assertional level). A DL terminology $\mathcal{T}$ (T-Box) is a set of **terminological axioms**, consisting of terminological constructions of ontological elements. Ontological elements are of two types : non-relational and relational elements. Non-relational elements are **concepts, datatypes, role characteristics** and **attribute characteristics**, and relational elements are **roles** and **attributes**. In the T-Box, axioms interpretations can be restricted to the models of $\mathcal{T}$. The A-Box describes properties of **instances** and **datavalues** of the domain. Also relations between them can be specified, they are called **assertional axioms** or **assertions**. An **interpretation** $\mathcal{I}$ is then a model for an ABox $\mathcal{A}$, if it is a model for all assertions. Finally, a DL knowledge base is a pair $K = (\mathcal{T}, \mathcal{A})$, and a model of K is an interpretation, which is both a model for $\mathcal{A}$ and $\mathcal{T}$. However, in the $\mathcal{SHOIN}(\mathcal{D})$ DL, distinction between T-Box and A-Box is not that obvious for all its constructs. Indeed, the $\mathcal{SHOIN}(\mathcal{D})$ DL introduces the notion of nominals (Horrocks & Patel-Schneider, 2003) corresponding to enumeration of instances I_i such as $\{I_1\} \sqcup \ldots \sqcup \{I_n\}$ *with* $i \in \mathbb{N}$, and role R and attribute A value restrictions such as $R{:}I_i$ and $A{:}V_i$ with $V_i \in sV$ and $i \in \mathbb{N}$. Axioms a_i built upon these constructs can be composed of both terminological elements and assertional elements. For example a concept D can be defined as the enumeration of n instances I_i by the axiom $a_j := D \equiv \{I_1\} \sqcup \ldots \sqcup \{I_n\}$. Consequently it cannot be identified as a terminological axiom nor an assertional axiom. That is why we will refer to the set of terminological axioms, assertional axioms and hybrid ones, simply as the **axioms of the knowledge base**. Also, for convenience, the $\mathcal{SHOIN}(\mathcal{D})$ ontology model proposed in this approach is designed to **model an entire $\mathcal{SHOIN}(\mathcal{D})$ knowledge base**, not only the ontology as such i.e. terminology.

The $\mathcal{SHOIN}(\mathcal{D})$ ontology model presented here is inspired by the Karlsruhe Ontology Model (Ehrig, 2004) and extended to cover the whole $\mathcal{SHOIN}(\mathcal{D})$ constructs. A $\mathcal{SHOIN}(\mathcal{D})$ ontology is a **set of axioms** (Haase & Stojanovic, 2005), which can be defined as a **mathematical structure**. Below is the definition of a structure according to (Marker, 2002):

Definition 1: A structure, is a n-uplet $\mathcal{S} = (\Omega, \Sigma, \Phi, E)$ consisting of:

i. A set Ω called the **underlying set** of $\mathcal{S}$.

ii. A collection Σ of **signature axioms** $\{\sigma_i: i \in I_1\}$ where $\sigma_i \subseteq \Omega^{m_i}$ for some $m_i \geq 1$.
iii. A collection Φ of **function axioms** $\{\varphi_i: i \in I_0\}$ where $\varphi_i: \Omega^{n_i} \to 2^{\Omega}$ for some $n_i \geq 1$.
iv. A collection E of **distinguished elements** $\{\varepsilon_i : i \in I_2\} \subseteq \Omega$.

Any (or all) of the sets I_0, I_1 and I_2 may be empty. We refer to n_i and m_j as the arity of φ_i and σ_j. For example, according to this definition, the structure defining the ordered field of real numbers has a domain $\mathbb{R}$, a signature $<$, binary functions $+, -, \times$, and distinguished elements 0 and 1, such that $\mathcal{S} = (\mathbb{R}, \{<\}, \{+, -, \times\}, \{0,1\})$.

2.2 Model definition of the structure for a $\mathcal{SHOIN}(\mathcal{D})$ Ontology

This part formalizes the $\mathcal{SHOIN}(\mathcal{D})$ ontology model as a mathematical fixed structure according to the structure definition (c.f. Definition 1).

Definition 2: **A $\mathcal{SHOIN}(\mathcal{D})$ ontology** is a structure $\mathcal{S}_O = (\Omega_O, \Sigma_O, \Phi_O, E_O)$ consisting of:

- The underlying set Ω_O containing:
  - Six disjoint sets sC, sT, sR, sA, sI, sV, sK_R and sK_A respectively called the **concept set**, the **datatype set**, the **role set**, the **attribute set**, the **instance set**, the **datavalue set**, the **role characteristic set** such as $sK_R = \{Symmetric, Functional, Inverse\ Functional, Transitive\}$ and the **attribute characteristic set** such as $sK_A = \{Functional\}$,
  - Four partial orders $\leq_C$, $\leq_T$, $\leq_R$ and $\leq_A$, respectively on sC called the **concept subsumption** or **taxonomy**, on sT called the **datatype subsumption**, on sR called the **role subsumption** and on sA called the **attribute subsumption**,

  such that $\Omega_O = \{(sC, \leq_C), (sT, \leq_T), (sR, \leq_R), (sA, \leq_A), sI, sV, sK_R, sK_A,\}$,

- The collection of signatures Σ_O containing the following sets of axioms:
  - A set of axioms $\sigma_R: sR \to sC^2$ called **role signature**,
  - A set of axioms $\sigma_A: sA \to sC \times sT$ called **attribute signature**,

  such that $\Sigma_O = \{\sigma_R, \sigma_A\}$

- The collection of functions Φ_O containing the following sets of axioms:
  - A set of axioms $\iota_C: sC \to 2^{sI}$ called **concept instantiation**,
  - A set of axioms $\iota_T: sT \to 2^{sV}$ called **datatype instantiation**,
  - A set of axioms $\iota_R: sR \to 2^{sI \times sI}$ called **role instantiation**,
  - A set of axioms $\iota_A: sA \to 2^{sI \times sV}$ called **attribute instantiation**,
  - A set of axioms $\kappa_R: sR \to 2^{sK_R}$ called **role characterization**,
  - A set of axioms $\kappa_A: sA \to 2^{sK_A}$ called **attribute characterization**,
  - A set of axioms $\varepsilon_C: sC \to 2^{sC}$ called **concept equivalence**,
  - A set of axioms $\varepsilon_R: sR \to 2^{sR}$ called **role equivalence**,
  - A set of axioms $\varepsilon_A: sA \to 2^{sA}$ called **attribute equivalence**,
  - A set of axioms $\varepsilon_I: sI \to 2^{sI}$ called **instance equivalence**,
  - A set of axioms $\delta_C: sC \to 2^{sC}$ called **concept disjunction**,
  - A set of axioms $\delta_I: sI \to 2^{sI}$ called **instance differentiation**,

- A set of axioms $-_C{:}sC \rightarrow 2^{sC}$ called **concept complement specification**,
- A set of axioms $-_R{:}sR \rightarrow 2^{sR}$ called **role inverse specification**,
- A set of axioms $maxCard_R{:}sR \rightarrow \mathbb{N}$called **role maximal cardinality restriction**,
- A set of axioms $minCard_R{:}sR \rightarrow \mathbb{N}$ called **role minimal cardinality restriction**,
- A set of axioms $\sqcap_C{:}sC \rightarrow 2^{sC}$ called **concept intersection**,
- A set of axioms $\sqcup_C{:}sC \rightarrow 2^{sC}$ called **concept union**,
- A set of axioms $\sqcup_I{:}sI \rightarrow 2^{sI}$ called **instance enumeration**,
- A set of axioms $\sqcup_V{:}sV \rightarrow 2^{sV}$ called **data value enumeration**,
- A set of axioms $\rho_{\exists R}{:}sR \rightarrow 2^{sC}$called **role existential restriction**,
- A set of axioms $\rho_{\forall R}{:}sR \rightarrow 2^{sC}$ called **role universal restriction**,
- A set of axioms $\rho_R{:}sR \rightarrow 2^{sI}$ called **role value restriction**,
- A set of axioms $\rho_{\exists A}{:}sA \rightarrow 2^{sT}$ called **attribute existential restriction**,
- A set of axioms $\rho_{\forall A}{:}sA \rightarrow 2^{sT}$ called **attribute universal restriction**,
- A set of axioms $\rho_A{:}sA \rightarrow 2^{sV}$ called **attribute value restriction**,

such that $\Phi_O=\{ \iota_C, \iota_T, \iota_R, \iota_A, \kappa_R, \kappa_A, \varepsilon_C, \varepsilon_R, \varepsilon_A, \varepsilon_I, \delta_C, \delta_I, -_C, -_R, maxCard_R, minCard_R, \sqcap_C, \sqcup_C, \sqcup_I, \sqcup_V, \rho_{\exists R}, \rho_{\forall R}, \rho_R, \rho_{\exists A}, \rho_{\forall A,}, \rho_A\}$.

- The collection of distinguished elements E_0 is composed of:

  - An element *TopConcept*, which is the special concept subsuming every concepts and instantiable with every instances,
  - An element *BottomConcept*, which is the special concept subsumed by every concepts and instantiable with no instances,
  - An element *TopAttribute,* which is the special attribute subsuming every attributes,
  - An element *BottomAttribute*, which is the special attribute subsumed by every attributes,
  - An element *TopRole*, which is the special role subsuming every roles,
  - An element *BottomRole*, which is the special role subsumed by every roles,
  - An element *TopDatatype*, which is the special datatype subsuming every datatypes,
  - An element *BottomDatatype,* which is the special datatype subsumed by every datatypes,

  such that $E_O=\{$ *TopConcept, BottomConcept, TopDatatype, BottomDatatype, TopRole, BottomRole, TopAttribute, BottomAttribute*$\}$

2.3 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Example

This part illustrates an application of the model on an example $\mathcal{SHOIN}(\mathcal{D})$ ontology inspired by the UOBM Ontology Benchmark (Ma & al, 2006). The example ontology, called $\mathcal{S}_O$, describes the relations between students taking courses, supervised by professors teaching courses. Instances and datavalues are added in order to show a complete illustration of the $\mathcal{SHOIN}(\mathcal{D})$ ontology model. A first DL version is given and is translated according to the correspondence table Table 5 (see Section 3):

- $\mathcal{S}_O$ as defined in Description Logics:
  - T-Box :

$$Person \sqsubseteq \top$$
$$HumanBeing \sqsubseteq \top$$
$$Student \sqsubseteq Person$$
$$Professor \sqsubseteq Person \sqcap \exists teaches.Course$$

$$Course \sqsubseteq \top \sqcap \exists duration.xsd{:}duration \sqcap \forall duration.xsd{:}time$$
$$KnowledgeCourse \sqsubseteq Course \sqcap isTaughtBy{:}christophe2 \sqcap duration{:}P2H$$
$$SemanticWebCourse \sqsubseteq Course$$
$$KnowledgeStudent \sqsubseteq Student \sqcap \forall takesCourse.KnowledgeCourse$$
$$Person \sqsubseteq NonStudent$$
$$xs{:}decimal \sqsubseteq \top$$
$$xs{:}string \sqsubseteq \top$$
$$xs{:}duration \sqsubseteq \top$$
$$friendOf \sqsubseteq \top$$
$$taughtBy \sqsubseteq \top$$
$$takesCourse \sqsubseteq \top$$
$$appliesTo \sqsubseteq \top$$
$$hasSupervisor \sqsubseteq \top$$
$$name \sqsubseteq \top$$
$$firstNameAndLastName \sqsubseteq \top$$
$$age \sqsubseteq \top$$
$$duration \sqsubseteq \top$$
$$Person \equiv HumanBeing$$
$$Course \equiv (KnowledgeManagementCourse \sqcup SemanticWebCourse) \sqcap isTaughtBy.$$
$$\leq 1\ Professor \sqcap isTaughtBy. \geq 1\ \top$$
$$takesCourse \equiv appliesTo$$
$$name \equiv firstNameAndLastName$$
$$christophe1 = cnicolle$$
$$Student \sqsubseteq \neg Professor$$
$$christophe1 \neq christophe2$$
$$Person \equiv NonStudent_{i}$$
$$teaches \equiv isTaughtBy_{i}$$
Person ⊑ friendOf.Person
Course ⊑taughtBy.Professor
Professor ⊑teaches.Course
Student⊑ hasSupervisor.Professor
Person ⊑name.xs :string
Person⊑ age.xs :decimal
Course⊑ duration.xs :duration
friendOf≡friendOf^{-}
$\top \sqsubseteq \leq age$
$\top \sqsubseteq \leq duration$

- A-Box :

Professor(christophe1)
Professor(christophe2)
Student(perrine)
Course(knowledgeManagement)
Course(knowledgeEngineering)
Course(semanticWeb1)
Course(semanticWeb2)
friendOf(christophe1, christophe2)
taughtBy(knowledgeManagement, christophe2)
teaches(chritophe1, semanticWeb1)
takesCourse(perrine, knowledgeManagement)

hasSupervisor(perrine, christophe1)
age(perrine, 26)
name(christophe1, « Christophe Nicolle »)

- $\mathcal{S}_O$ as defined by our model:
  - Definition of Ω_O={(sC, $\leq_C$),(sT, $\leq_T$),(sR, $\leq_R$),(sA, $\leq_A$), sI, sV, sK_R, sK_A,}:

  - *sC ={TopConcept, Person, HumanBeing, Student, Professor, Course, KnowledgeCourse, SemanticWebCourse, KnowledgeStudent, NonStudent},*
  - $\leq_C$ ={(TopConcept, Person), (TopConcept, HumanBeing), (Person, Student), ($\sqcap_C$($\rho_{\exists R}$ (teaches, Course), Person), Professor), ($\sqcap_C$($\rho_{\exists A}$ (duration, xsd:duration), TopConcept), Course), ($\rho_{\forall A}$ (duration, xsd: time), TopConcept)), ($\sqcap_C$(Course, $\sqcap_C$(ρ_R (isTaughtBy, christophe2), ρ_A (duration, P2H)), KnowledgeCourse), (Course, SemanticWebCourse), (Student, KnowledgeStudent), (Person, NonStudent),($\sqcap_C$(Student, $\rho_{\forall R}$ (takesCourse, KnowledgeCourse)), KnowledgeStudent)};*

  - *sT ={TopDataType, xs:decimal, xs:string, xs:duration},*
  - $\leq_T$ ={(TopDataType, xs:decimal), (TopDataType, xs:string), (TopDataType, xs:duration)},*

  - *sR ={TopRelation, friendOf, taughtBy, teaches, takesCourse, appliesTo, hasSupervisor},*
  - $\leq_R$ ={(TopRelation, friendOf), (TopRelation, taughtBy), (TopRelation, takesCourse), (TopRelation, appliesTo), (TopRelation, hasSupervisor)},*

  - *sA ={ TopAttribute, name, firstNameAndLastName, age, duration},*
  - $\leq_A$ ={(TopAttribute, name), (name, firstNameAndLastName), (TopAttribute, age), (TopAttribute, duration)},*

  - *sI ={christophe1, cnicolle, christophe2, perrine, knowledgeManagement, knowledgeEngineering, semanticWeb1, semanticWeb2},*

  - *sV ={"Christophe Nicolle", "Christophe Cruz", "Perrine Pittet", 26, 26.0, P2H, P4H},*

  - sK_R ={Symmetric, Functional, Inverse Functional, Transitive},*

  - sK_A ={Functional},*

  - Definition of Σ_O={σ_R, σ_A}:

  - σ_R={(takesCourse, (Person, Course)), (friendOf, (Person, Person)), (taughtBy, (Course, Professor)), (teaches, (Professor, Course)), (hasSupervisor, (Student, Professor))};
  - σ_A={(name, (Person, xs:string)), (age, (Person, xs:decimal)), (duration, (Course, xs:duration))};

    - Definition of Φ_O={ι_C, ι_T, ι_R, ι_A, κ_R, κ_A, ε_C, ε_R, ε_A, ε_I, δ_C, δ_I, $-_C$, $-_R$, $maxCard_R$, $minCard_R$, $\sqcap_C$, $\sqcup_C$, $\sqcup_I$, $\sqcup_V$, $\rho_{\exists R}$, $\rho_{\forall R}$, $\rho_{\forall R}$, ρ_R, $\rho_{\exists A}$, $\rho_{\forall A}$, ρ_A}

- ι_C={(Professor, christophe1), (Professor, christophe2), (Student, perrine), (Course, knowledgeManagement), (Course, knowledgeEngineering), (Course, semanticWeb1), (Course, semanticWeb2)};
- ι_T={(xsd:decimal,26), (xsd:string, "Christophe Nicolle"), (xsd:string, "Christophe Cruz"), (xsd:string, "Perrine Pittet"), (xsd:duration,P2H), (xsd:duration, P4H)};
- ι_R={(friendOf,(christophe1, christophe2)), (taughtBy(knowledgeManagement, christophe2)), (teaches, (christophe1, semanticWeb1)), (takesCourse, (perrine, knowledgeManagement)), hasSupervisor(perrine, christophe1)}};
- ι_A={(age,(perrine, 26)), (name,(christophe1, "Christophe Nicolle")), (name,(christophe2, "Christophe Cruz)), (name,(perrine, "Perrine Pittet")), (duration, (knowledgeManagement, P2H))};
- κ_R={(friendOf, Symmetric), (taughtBy, Functional), (teaches, InverseFunctional)};
- κ_A={(age, Functional), (duration, Functional)};
- ε_C={(Person, HumanBeing), (Person, $\sqcap_C$(Student, NonStudent)), (Course, $\sqcap_C$($\sqcup_C$(KnowledgeCourse, SemanticWebCourse), $\sqcap_C$(maxCardR (isTaughtBy, 1), minCardR(isTaughtBy, 1))};
- ε_R ={(takesCourse, appliesTo)};
- ε_A={(name, firstNameAndLastName)};
- ε_I={(christophe1, cnicolle)};
- δ_C={(Student, Professor)};
- δ_I={(christophe1,christophe2)};
- -$_C$={(Student, NonStudent)};
- -$_R$={(teaches, isTaughtBy)};
- maxCardR={(isTaughtBy, 1)};
- minCardR={(isTaughtBy, 1)};
- $\sqcap_C$= {{Student, NonStudent}, ($\sqcup_C$(KnowledgeCourse, SemanticWebCourse), $\sqcap_C$(maxCardR (isTaughtBy, 1), minCardR(isTaughtBy, 1)), (Person, $\rho_{\exists R}$ (teaches, Course)), (Student, $\rho_{\forall R}$(takesCourse, KnowledgeCourse)), ($\rho_{\exists A}$(duration, xsd:duration), TopConcept), ($\rho_{\forall A}$(duration, xsd: time), TopConcept), $\sqcap_C$($\rho_{\exists A}$(duration, xsd:duration), TopConcept), ($\rho_{\forall A}$(duration, xsd: time), TopConcept)), (ρ_R(isTaughtBy, christophe2), ρ_A(duration, P2H)), (Course, $\sqcap_C$(ρ_R(isTaughtBy, christophe2), ρ_A(duration, P2H))))};

- $\sqsubset_C$={{KnowledgeCourse, SemanticWebCourse}};
- $\sqsubset_I$={(knowledgeManagement, knowledgeEngineering)};
- $\sqsubset_V$={{26.0, 26.00}};
- $\rho_{\exists R}$= {(teaches, Course), (takesCourse, Course)};
- $\rho_{\forall R}$={(takesCourse, KnowledgeCourse)};
- ρ_R={(isTaughtBy, christophe2)};
- $\rho_{\exists A}$ ={ (duration, xsd:duration)};
- $\rho_{\forall A}$={(duration, xsd: time)};
- ρ_A ={(duration, P2H)}.

  - Definition of E_0: E_0 = {TopConcept, BottomConcept, TopAttribute, BottomAttribute, TopRole, BottomRole, TopDataType, BottomDataType}

Figure 15 shows a graphical representation of $\mathcal{S}_0$ realized with the G-MOT Ontology Editor (Paquette & Magnan, 2008).

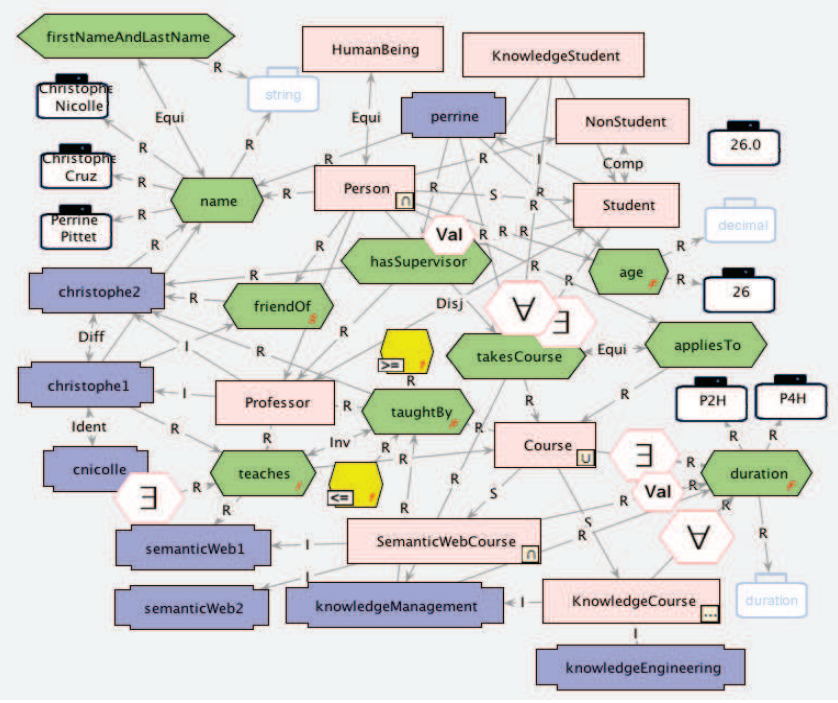


Figure 15. Graphical Representation of the Ontology $\mathcal{S}$O with the G-MOT Editor.

3 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Change Modelling

This section formalizes the change modelling activity according to the proposed model. The first part gives definitions about ontological changes, such as the definition of a change, of basic and composed changes, ontology log of changes and global log of changes. The second part presents the list of basic changes tailored from the $\mathcal{SHOIN}(\mathcal{D})$ allowed constructs and their representation in evolution logs. The third part presents an approach to model composed changes from basic ones and their representation in evolution logs.

3.1 Change Definitions

This part gives some definitions about changes, such as the definition of a change, of basic and composed changes, ontology log of changes and global log of changes. A change modelling example is given.

Definition 3: A change β is the application of an addition or a deletion of an axiom a_i on an ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$. It is defined by an operator π and a set of parameters ε_n with $n \in \mathbb{N}$ and $\varepsilon_n \in \{sC, sT, sR, sA, sI, sV\} \subseteq \Omega_0$. A change potentially adds or deletes one or more axioms in Ω_0, Σ_0, and Φ_0.

Like in (Klein, 2004) two change types are distinguished, basic and composed changes.

Definition 4: A basic change is a n-tuple $\beta_{Bi}=(\pi, (\varepsilon_1, ..., \varepsilon_n))$, $\pi \in \Pi$. The set Π of all operators is composed of the addition and deletion operators π_i that respectively add and delete the axioms a_i in the sets corresponding to their type, among Ω_0, Σ_0, and Φ_0. If π_i is an addition operator, then $\beta_{Bi} \rightarrow \mathcal{S}_0 + a_i$. If π_I is a deletion operator, then $\beta_B \rightarrow \mathcal{S}_0 - a_i$.

Definition 5: A composed change is a n-tuple $\beta_C = (\beta_{B_1}, ..., \beta_{B_n})$ of basic changes β_{B_i}.

The definitions of a log of changes and an ontology global log of changes can be therefore formalized. The first one is used to represent changes to apply on an ontology. The second is the concatenation of the changes issued from the several ontology evolutions and allows to trace changes applied on it since its creation.

Definition 6: A log of changes is defined by a n-tuple $log_i = (\beta_1, ..., \beta_n)$ composed of basic and composed changes β_i.

Definition 7: The global log of a given an ontology $\mathcal{S}_0 = (\Omega_0, \Sigma_0, \Phi_0, E_0)$, noted log_0, is defined by an n-tuple $log_0 = (log_1, ..., log_n)$ of logs of changes log_i.

To illustrate these definitions, here is an example of a global log of changes log_o composed of two logs of changes log_1 and log_2, which are respectively composed of two basic changes $\beta_{B_1}=(\pi_1, (\varepsilon_1, ..., \varepsilon_n))$, and $\beta_{B_2}=(\pi_2, (\varepsilon_1, ..., \varepsilon_m))$, and one composed change $\beta_{C_3}=(\beta_{B_3}, \beta_{B_4}, \beta_{B_5})$. The log of changes log_1 and log_2 are defined by $log_1 = (\beta_{B_1}, \beta_{B_2})$ *and* $log_2=(\beta_{C_3})=(\beta_{B_3}, \beta_{B_4}, \beta_{B_5})$. The global log of changes is defined by $log_o = (log_1, log_2)$.

3.2 $\mathcal{SHOIN}(\mathcal{D})$ Basic Changes Representation

This part presents the list of basic changes tailored from the $\mathcal{SHOIN}(\mathcal{D})$ allowed constructs and their representation in evolution logs. To produce the list of basic change operator types on ontologies, the $\mathcal{SHOIN}(\mathcal{D})$ model is exploited as described in Table 5. The third column lists the **84 operators** representing basic changes. According to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model, every basic change can be declined as an addition or a deletion of an axiom of the underlying set Ω_O, the set of signatures Σ_O or the set of functions Φ_O.

	$\mathcal{SHOIN}(\mathcal{D})$ **DL Syntax**	$\mathcal{SHOIN}(\mathcal{D})$ **Sets**	**Basic Change Types** π	**Parameter Types** $\mathcal{E}_1, \ldots, \mathcal{E}_n$
Descriptions	C	sC	addConcept deleteConcept	(Concept)
	$C_1 \sqcap \ldots \sqcap C_n$	$\sqcap_C$	**addConceptIntersection** **deleteConceptIntersection**	**(Concept$_1$,...,Concept$_n$)**
	$C_1 \sqcup \ldots \sqcup C_n$	$\sqcup_C$	**addConceptUnion** **deleteConceptUnion**	**(Concept$_1$,...,Concept$_n$)**
	$\neg C$	$-_C$	addComplementConcept deleteComplementConcept	(Concept)
	$\{I_1\} \sqcup \ldots \sqcup \{I_n\}$	$\sqcup_{IC}$	**addInstanceEnumeration** **deleteInstanceEnumeration**	**(Instance$_1$,...,Instance$_n$)**
	$\exists R.C$	$\rho_{\exists R}$	addRoleExistentialRestriction deleteRoleExistentialRestriction	(Role, Concept)
	$\forall R.C$	$\rho_{\forall R}$	addRoleUniversalRestriction deleteRoleUniversalRestriction	(Role, Concept)
	$R : I$	$\rho_{R:}$	addRoleValueRestriction deleteRoleValueRestriction	(Role, Instance)
	$\geq n\ R$	$maxCard_R$	addRoleMinCardinality deleteRoleMinCardinality	(Role, n)
	$\leq n\ R$	$minCard_R$	addRoleMaxCardinality deleteRoleMaxCardinality	(Role, n)
	$\exists A.T$	$\rho_{\exists A}$	addAttributeExistentialRestriction deleteAttributeExistentialRestriction	(Attribute, Datatype)
	$\forall A.T$	$\rho_{\forall A}$	addAttributeUniversalRestriction deleteAttributeUniversalRestriction	(Attribute, Datatype)
	$A : V$	$\rho_{A:}$	addAttributeValueRestriction deleteAttributeValueRestriction	(Attribute, Datavalue)
	T	sT	addDatatype deleteDatatype	(Datatype)
	$\{V_1\} \sqcup \ldots \sqcup \{V_n\}$	$\sqcup_V$	**addDatavalueEnumeration** **deleteDatavalueEnumeration**	**(Datavalue$_1$,...,Datavalue$_n$)**
	R R^-	sR	addRole deleteRole	(Role)
	A	sA	addAttribute deleteAttribute	(Attribute)
	I	sI	addInstance deleteInstance	(Instance)
	V	sV	addDatavalue deleteDatavalue	(Datavalue)
	$C_1 \sqsubseteq C_2$	$\leq_C$	addSubConcept deleteSubConcept	(Concept$_1$, Concept$_2$)
	$C_1 \equiv \ldots \equiv C_n$	ε_C	**addEquivalentConcept** **deleteEquivalentConcept**	**(Concept$_1$,...Concept$_n$)**
	$\bot \equiv C_1 \sqcap C_2$	δ_C	addDisjointConcept deleteDisjointConcept	(Concept$_1$, Concept$_2$)
	$T_1 \sqsubseteq T_2$	$\leq_T$	addSubDatatype deleteSubDatatype	(Datatype$_1$, Datatype$_2$)
	$V \in T_i$	ι_T	addDatatypeInstanciation deleteDatatypeInstanciation	(Datavalue, Datatype)
	$R \sqsubseteq R_i$	R	addRole deleteRole	(Role)
	$\geq 1R \sqsubseteq C_i$ $\top \sqsubseteq \forall R.C_i$	σ_R	addRoleSignature deleteRoleSignature	(Role, Concept,Concept)

$R \equiv R_0^-$	$-_R$	addInverseRole deleteInverseRole	(Role$_1$ Role$_2$)
$R \equiv R^-$	κ_R	addSymmetricRoleProperty deleteSymmetricRoleProperty	(Role)
$\top \sqsubseteq \leq 1R$		addFunctionalRoleProperty deleteFunctionalRoleProperty	(Role)
$\top \sqsubseteq \leq 1R^-$		addInverseFunctionalRoleProperty deleteInverseFunctionalRoleProperty	(Role)
$Tr(R)$		addTransitiveRoleProperty deleteTransitiveRoleProperty	(Role)
$R_1 \sqsubseteq R_2$	$\leq_R$	addSubRole deleteSubRole	(Role$_1$, Role$_2$)
$R_1 \equiv \ldots \equiv R_n$	ε_R	**addEquivalentRole** **deleteEquivalentRole**	**(Role$_1$,...,Role$_n$)**
$A \sqsubseteq A_i$	A	addAttribute deleteAttribute	(Attribute)
$\geq 1A \sqsubseteq C_i$ $\top \sqsubseteq A.T_i$	σ_A	addAttributeSignature deleteAttributeSignature	(Attribute Concept,Datatype)
$\top \sqsubseteq \leq 1A$	κ_A	addFunctionalAttrbuteProperty deleteFunctionalAttrbuteProperty	(Attribute)
$A_1 \sqsubseteq A_2$	$\leq_A$	addSubAttribute deleteSubAttribute	(Attribute$_1$, Attribute$_2$)
$A_1 \equiv \ldots \equiv A_n$	ε_A	**addEquivalentAttribute** **deleteEquivalentAttribute**	**(Attribute$_1$,...,Attribute$_n$)**
$I \in C_i$	ι_C	addConceptInstanciation deleteConceptInstanciation	(Instance, Concept)
$\{I, I_j\} \in R_i$	ι_R	addRoleInstanciation deleteRoleInstanciation	(Instance, Instance$_1$, Role)
$\{I, V_j\} \in A_i$	ι_A	adAttributeInstanciation deleteAttributeInstanciation	(Instance, Datavalue, Attribute)
$\{I_1\} \equiv \ldots \equiv \{I_n\}$	ε_I	**addSameInstance** **deleteSameInstance**	**(Instance$_1$,...,Instance$_n$)**
$\{I_i\} \sqsubseteq \neg\{I_j\}, i \neq j$	δ_I	**addDifferentInstancce** **deleteDifferentInstance**	**(Instance$_1$,...,Instance$_n$)**

Table 5. The list of $\mathcal{SHOIN}(\mathcal{D})$ Basic Change Operator Types and their Parameters.

To illustrate the basic change modelling according to the Table 5 basic change types list, here is an example of basic change β_{B_1} using the basic change operator type π_1= *addConcept*, with a parameter ε_1=*Concept1*. This basic change, adding the concept Concept1 to the concept set *sC* of an ontology $\mathcal{S}_0$ is defined by β_{B_1}=(π_1, (ε_1))=(*addConcept, (Concept1)).*

18 of the 84 change types (appearing in bold on Table 5) can potentially turn into composed changes if applied on more than two elements of the ontology. For instance, the change operator type *addConceptIntersection,* which application syntax is *addConceptIntersection(Concept$_1$,...,Concept$_n$)* is considered basic if used to add the intersection of two concepts (ex: *addConceptIntersection (Concept$_1$, Concept$_2$))*, but composed if used to add the intersection of more than two concepts (ex: *addConceptIntersection(Concept$_1$, Concept$_2$, Concept$_3$)*. Actually, in the second case, its application is composed of several basic intersections of concepts (ex: *addConceptIntersection(addConceptIntersection (Concept$_1$, Concept$_2$), Concept$_3$)).*

3.3 $\mathcal{SHOIN}(\mathcal{D})$ Composed Changes Representation

This part presents an empirical approach to model composed changes according to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model.

An **infinite set of composed changes** can be generated from the aggregation of basic changes (Plessers & De Troyer, 2005). Their relevance depends on the need of particular changes implied by **particular uses**. For instance, the concept renaming change - as the set-theory change not the lexical one - is often used in collaborative development of an ontology when reaching a consensus on a concept name but can be useless in other contexts. For this reason, the proposed model natively provides the limited set

of 84 basic change operator types but, depending on change modelling needs, gives the opportunity to build composed change operator types from these basic ones.

For instance, the composed change type β_{C_0} called *addConceptAsConceptUnion* defined by the addition of a concept Concept defined as equivalent to the union of two concepts *Concept1* and *Concept2*, noted $Concept \equiv Concept1 \sqcup Concept2$ in DL, can be modelled by the following composition of basic changes:

β_{C_0}*=(AddConceptAsConceptUnion, (Concept, Concept1, Concept2))*
=(
(addConceptUnion,(Concept1, Concept2)),
(addConcept,(Concept)),
(addConceptEquivalence,(Concept, $\sqcup_C$*(Concept1, Concept2))*
)

The first basic change *addConceptUnion,(Concept1, Concept2)* adds a new concept union between *Concept1* and *Concept2* to the subset $\sqcup_C$, such as $\sqcup_C$*(Concept1, Concept2)* can be modelled. The second basic change *addConcept,(Concept)* adds the concept *Concept* to the subset *sC.* The third basic change *addConceptEquivalence,(Concept,* $\sqcup_C$ *(Concept1, Concept2))* adds a new concept equivalence between *Concept* and the concept union previously added $\sqcup_C$(Concept1, Concept2)) to the subset $\mathcal{E}_C$ such that $\mathcal{E}_C$*(Concept,* $\sqcup_C$*(Concept1, Concept2))* can be modelled.

The impact of the application of β_{C_0} is defined by the impact of each basic change composing β_{C_0} on their corresponding ontology subsets (see Table 5). On an ontology $\mathcal{S}_O$*=(*Ω_O*,* Σ_O*,* Φ_O*,* E_O*)* evolving in $\mathcal{S}_{Onew}$ this impact can be modelled as:

sC_{new}*=sC+{Concept};*
$\sqcup_{C\,new}$=$\sqcup_C$*+{(Concept1, Concept2)};*
$\mathcal{E}_{C\,new}$=$\mathcal{E}_C$*+{(Concept,* $\sqcup_C$*(Concept1, Concept2))}.*

The corresponding composed deletion change type β_{C_1} called *DeleteConceptAsConceptUnion removing the* concept Concept defined as equivalent to the union of two concepts *Concept1* and *Concept2*, can be modelled as follows:

β_{C_1}*=(DeleteConceptAsConceptUnion, (Concept, Concept1, Concept2))*
=(
(deleteConceptEquivalence,(Concept, $\sqcup_C$*(Concept1, Concept2)),*
(deleteConcept,(Concept)),
(deleteConceptUnion,(Concept1, Concept2))
)

The first basic change *deleteConceptEquivalence,(Concept,* $\sqcup_C$*(Concept1, Concept2))* removes the concept equivalence between *Concept* and the concept union $\sqcup_C$(Concept1, Concept2)) from the subset $\mathcal{E}_C$. The second basic change *deleteConcept,(Concept)* removes the concept *Concept* from the subset *sC.* The third basic change *deleteConceptUnion,(Concept1, Concept2)* removes the concept union between *Concept1* and *Concept2* from the subset $\sqcup_C$.

The impact of the application of β_{C_1} on an ontology $\mathcal{S}_O$*=(*Ω_O*,* Σ_O*,* Φ_O*,* E_O*)* evolving in $\mathcal{S}_{Onew}$ can be modelled as:

sC_{new}*=sC-{Concept};*
$\sqcup_{C\,new}$=$\sqcup_C$*-{(Concept1, Concept2)};*
$\mathcal{E}_{C\,new}$=$\mathcal{E}_C$*-{(Concept,* $\sqcup_C$*(Concept1, Concept2))}.*

4 Conclusion

This chapter has presented the first part of the first block of my approach consisting in the definition of a collaborative change management process ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology. The **collaborative change management methodology** has been detailed according to the six phases of the ontology evolution process. A global log of changes, tracing all the changes applied on an ontology since its creation is updated after each validated phase of the ontology evolution process in order to display the versioning features of the methodology and ensure change semantics structural and logical consistency qualification, view sub-ontology extraction and change propagation impact management. **The $\mathcal{SHOIN}(\mathcal{D})$ ontology model** required for the change modelling phase has been formalized. Its fixed structure allows modelling structurally consistent changes by guarantying the expressivity level of the ontology after their application. However it is not sufficient to maintain the structural consistency of the ontology. An additional **structural consistency preventive approach** based on axiom dependencies resolution is performed at the end of the change modelling phase. Also a **preventive logical consistency qualification** of these changes is included at the change semantics phase in order to help the user ensure their logically consistent application. These two structural and logical consistency preventive approaches are formalized in the next chapter to complete the first block of my approach.

References

Baader, F. (Ed.). (2003). The description logic handbook: theory, implementation, and applications. Cambridge university press.

Djedidi, R. (2009). Approche d'évolution d'ontologie guidée par des patrons de gestion de changement (Doctoral dissertation, Université Paris Sud-Paris XI).

Ehrig, M. H. (2004). Similarity for ontologies-a comprehensive framework. Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM.

Haase, P., & Stojanovic, L. (2005). Consistent evolution of OWL ontologies. In The Semantic Web: Research and Applications (pp. 182-197). Springer Berlin Heidelberg.

Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In The Semantic Web-ISWC 2003 (pp. 17-29). Springer Berlin Heidelberg.

Klein, M. C. A. (2004). Change management for distributed ontologies. PhD Thesis.

Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., & Liu, S. (2006). Towards a complete OWL ontology benchmark. In The Semantic Web: Research and Applications (pp. 125-139). Springer Berlin Heidelberg.

Marker, D. (2002). Model theory: an introduction, volume 217 of Graduate Texts in Mathematics. Springer-Verlag, New York, 6, 31.

Paquette, G., & Magnan, F. (2008). An executable model for virtual campus environments. In Handbook on Information Technologies for Education and Training (pp. 363-403). Springer Berlin Heidelberg.

Plessers, P., & De Troyer, O. (2005). Ontology change detection using a version log. In The Semantic Web–ISWC 2005 (pp. 578-592). Springer Berlin Heidelberg.

Stojanovic, L. (2004). Methods and tools for ontology evolution. PhD Thesis.

Chapter 5

$\mathcal{SHOIN}(\mathcal{D})$ Structural and Logical Consistency Preventive Resolution

Summary

This chapter presents the **structural and logical preventive resolution approaches** inspired and based on the $\mathcal{SHOIN}(\mathcal{D})$ model presented in Chapter 4. A first section introduces a $\mathcal{SHOIN}(\mathcal{D})$ ontology consistency maintenance background and basic definitions of its structural and logical subtypes. A second section formalizes the structural consistency preventive resolution approach used in the change management methodology during the modelling phase. It is based on **change structural dependencies** and automatically resolves structurally inconsistent changes represented by a user. A third section formalizes the logical consistency preventive resolution approach used to help the user identifying and debugging **minimal logically inconsistent sets of changes** modelled, before implementing them on the ontology. It is based on **axiom unsatisfiability constraints** derived from $\mathcal{SHOIN}(\mathcal{D})$ changes logical constraints.

Plan

1 Introduction on SHOIN(D) Ontology Consistency 115
2 Structural Consistency Preventive Approach 117
2.1 SHOIN(D) Structural Constraints 117
2.2 Structural Consistency Resolution Example 118
2.3 Discussion 119
3 Logical Consistency Preventive Approach 121
3.1 SHOIN(D) Inconsistency 121
3.2 SHOIN(D) Logical Inconsistency Preventive Approach 123
3.2.1 Basic SHOIN(D) Change Unsatisfiability Constraints 123
3.2.2 Composed SHOIN(D) Change Unsatisfiability Constraints 125
3.2.3 Logical Consistency Preventive Approach Process 126
3.3 Logical Consistency Preventive Analysis Example 127
3.3.1 Logical Consistency Detection Example 128
• 132
3.3.2 Logical Consistency Resolution Example 140
3.4 Discussion 140
4 Conclusion 141

Previous chapter has presented the first part of the first block composing the approach of my thesis. This chapter completes it by the **formalization of the structural and logical consistency preventive approaches required for the change modelling and change semantics phases** of the evolution process of the OntoVersionGraph methodology. The structural approach is used to **automatically resolve structural inconsistencies of the list of changes modelled** by domain experts during the change modelling phase. It aims at providing a **structural consistent log of changes** for the change semantics phase. The logical approach **automatically checks the log to identify minimal sets of changes potentially causing logical inconsistencies** in order to guide the domain experts for the logical consistency resolution. Its role is to help the domain experts provide a logical consistent log of changes for the change implementation phase. The first section brings the preliminary definitions on which the two approaches are based. The two approaches are respectively formalized in the second and the third sections.

1 Introduction on $\mathcal{SHOIN}(\mathcal{D})$ Ontology Consistency

This section brings the preliminary definitions at the basis of the two consistency resolution approaches presented in this chapter.

According to (Rogozan, 2009), an ontology is considered consistent if all the constraints associated with the ontology are met. Maintaining consistency during ontology evolution therefore first implies **studying the various constraints that must meet each change** to ensure its consistent application on the ontology, and second **modelling changes compliant with these constraints**. As seen in Chapter 3, the main constraints are related to the **syntax** (model) or the **logic** (axioms) of the language. They respectively correspond to the **structural consistency** and **logical consistency** of the ontology. According to (Horrocks & Patel-Schneider, 2003), maintaining structural consistency implies modeling **respectful changes according to the ontology language constructs** (e.g. conforming use of the concept, of property, etc.). Logical consistency addresses the question of whether the ontology is "semantically correct", i.e. contains no contradictory information between the ontology axioms. Maintaining logical consistency implies modelling **changes compliant with the logical constraints described by the ontology axioms**. In the OntoVersionGraph methodology, an **ontology is assumed consistent** if it is structurally and logically consistent. A $\mathcal{SHOIN}(\mathcal{D})$ consistent ontology can therefore be defined as below.

Definition 8: A $\mathcal{SHOIN}(\mathcal{D})$ Consistent Ontology is a structurally and logically consistent ontology, i.e. compliant with the $\mathcal{SHOIN}(\mathcal{D})$ model constructs and the logical constraints described by its axioms.

As seen in Chapter 3, two main consistency maintenance approach types are identified in the literature to maintain ontology consistency: **corrective and preventive resolution approaches**. Corrective approaches consist in the **evaluation of the ontology consistency after changes are applied** on it, whereas preventive approaches **assess the changes impact on the ontology consistency before they are applied** on it. Both approaches can be used during the evolution process. The use of corrective approaches is however heavier in terms of resources, as consistency checking is done directly on the ontology. Most of reasoners, for instance, have to load it entirely to parse it according to Tableaux-Calculus algorithms. Yet DL reasoners have proven their efficiency in terms of inconsistency detection (Pan, 2005) and can be used to reinforce the reliability of logical consistency resolution methodologies.

As stated in Chapter 4, the **Pellet reasoner** is used at the end of the change implementation phase as a **logical consistency corrective resolution** approach to afford a logical consistency additional check. Pellet, yet incomplete for $\mathcal{SHOIN}(\mathcal{D})$ (with A-Boxes), is claimed to be sound and complete on $\mathcal{SHIN}(\mathcal{D})$ and $\mathcal{SHON}(\mathcal{D})$ DLs. It is considered as the first reasoner that supported all of $\mathcal{SHOIN}(\mathcal{D})$ in (Parsia & Sirin, 2004) and has been extended to $\mathcal{SROIQ}(\mathcal{D})$ (equivalent to OWL2 expressivity) . Concerning

structural consistency, it is ensured by both the $\mathcal{SHOIN}(\mathcal{D})$ ontology model, providing a limited set of changes, and a complementary structural consistency preventive resolution. Therefore changes cannot be misused regarding the structural constructs of the $\mathcal{SHOIN}(\mathcal{D})$ language, and their application is assessed as non-harming for the ontology structure. A **structural consistency corrective approach is consequently not embedded** in OntoVersionGraph.

This chapter therefore formalizes two **structural and logical consistency preventive approaches**. Both are based on the $\mathcal{SHOIN}(\mathcal{D})$ ontology model defined in Chapter 4. The structural consistency preventive approach is addressed in the second section. The logical consistency preventive approach is presented in the third section.

2 Structural Consistency Preventive Approach

This section focuses on the **structural consistency preventive resolution approach** used in the OntoVersionGraph methodology during the modeling phase of the ontology evolution process.

The structural consistency preventive resolution approach intervenes at the change modelling phase, after domain experts have chosen the changes to apply on the ontology among the $\mathcal{SHOIN}(\mathcal{D})$ ontology model changes list. The corresponding **log of changes is used by this approach to detect structurally non-compliant changes.** As the basic changes list derived from the $\mathcal{SHOIN}(\mathcal{D})$ ontology model is limited and syntactically consistent, they cannot be misused by the user, who selects them. However, **deletion changes** which are used to delete ontological elements from an ontology need to be applied with other depending changes to maintain the ontology structural consistency.

To illustrate the issue, let us consider an ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$ in which is represented the axiom $a1:=\{\rho_{\exists R}(hasJobType)=(hasJobType, (Professor, TemporaryTeacher))\}$ and a log of changes in which is represented the change $\beta_{B1}=(deleteConcept,(Professor))$, which corresponds to the deletion of the axiom $a0:=\{Professor \in sC\}$. The deletion of the concept *Professor* cannot be realised as a standalone operation because of the existence of the axiom a_1 in the ontology, which would become structurally incomplete. Moreover, the ontology modified with such a deletion becomes structurally inconsistent. A revision strategy has to be chosen to maintain the structural consistency.

In this methodology we have chosen to remove the axioms dependent to the ontological element that has to be deleted in order to automatize the process in the simplest way. Considering our example, if the concept *Professor* has to be removed, so has to be the axiom a_1, as such as any axiom of the ontology concerning this concept. And it has to be done simultaneously or, following the order of the concept **dependencies**, by first deleting the axioms of the function set Φ_0 having this concept as parameter, then the role and attribute signatures of the signature set Σ_0 having this concept as domain or range, then the partial orders relations of this concept defined in Ω_0, and finally the concept itself. These dependencies are **structural constraints**, which respect the following axiom dependency definition.

Definition 9: Axiom Dependency. Let $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$ be a $\mathcal{SHOIN}(\mathcal{D})$ ontology, if an ontological element $\varepsilon \in \{sC, sT, sR, sA, sI, sV\} \subseteq \Omega_0$ is reused by an axiom $a \in \mathcal{S}_0$, then a depends on ε and has to be deleted before or simultaneously with a .

2.1 $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraints

This first part defines the **structural constraints** based on the axiom dependency property (c.f. Prop. 1) that conditions the application structural consistency of deletion changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology. The definition of $\mathcal{SHOIN}(\mathcal{D})$ structural constraints is given below.

Definition 10: $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraint. A structural constraint applying to the deletion of an ontological element $\varepsilon \in \{sC, sT, sR, sA, sI, sV\} \subseteq \Omega_0$ defines a dependency between ε and the an axiom $a \in \mathcal{S}_0$ if ε is a parameter of a.

According to this definition, Table 6 shows the axiom dependencies between $\mathcal{SHOIN}(\mathcal{D})$ ontological elements types (Concept, Datatype, Relation, Attribute, Instance, Datavalue) and each axiom subset of the underlying set, the signature set and the function set of a $\mathcal{SHOIN}(\mathcal{D})$ ontology, i.e. the whole axiom subsets of the ontology, as a **dependency matrix**.

	sC	sT	sR	sA	sI	sV	sKR	sKA	$\forall\sqsubseteq_C$	$\forall\sqsubseteq_T$	$\forall\sqsubseteq_R$	$\forall\sqsubseteq_I$	σ_R	σ_A	ι_C	ι_T	ι_R	ι_A	κ_R	κ_A	ε_C	ε_R	ε_A	ε_I	δ_C	δ_I	$\neg_C$	$\neg_R$	$maxCard_R$	$minCard_R$	$\sqcap_C$	$\sqcup_C$	$\sqcup_i$	$\sqcup_V$	$\rho_{\exists R}$	$\rho_{\forall R}$	ρ_R	$\rho_{\exists A}$	$\rho_{\forall A}$	ρ_A
Concept	x								x				x	x	x						x				x		x				x	x			x	x				
Datatype		x								x				x		x																					x	x	
Role			x				x				x		x				x		x			x						x	x	x					x	x	x			
Attribute				x				x				x		x				x		x			x														x	x	x
Instance					x										x		x	x						x		x						x							
Datavalue						x										x		x														x							x

Table 6. Structural Dependencies Between Ontological Elements and Axiom Subsets

The value *x*, represented by *dependency[i][j]=x*, indicates that there is a dependency between the element type of the row *i* and the axiom subset of the column *j*. For instance *dependency[13][1]=x* indicates that there is a structural dependency between a concept C_i (c.f. Concept in *dependency[0][1]*) and all the axioms a_i of the role signature subset σ_R (c.f. σ_R in *dependency[13][0]*), which take C_i as domain or range. It means that if the deletion change $\beta_1=(deleteConcept, (C_i))$, is modelled in the log of changes during the change modelling phase, the deletion changes of the form $\beta_{Bj}=(deleteRoleSignature, (C_i, D))$ or $\beta_{Bj}=(deleteRoleSignature, (D, C_i))$, with *D* an arbitrary concept, have to be added to the log of changes.

2.2 Structural Consistency Resolution Example

This part illustrates the structural consistency preventive resolution approach with a simple example. Let us consider the following structurally consistent set of DL axioms defining an ontology $\mathcal{S}_0$, where *Person*, *Student* and *Professor* are defined concept names:

$$Student \sqcap Professor \equiv \bot$$
$$Professor \equiv Person \sqcap \exists teaches.T$$

Concepts *Student* and *Professor* are defined as disjoint concepts. The existential restriction $\exists teaches.T$ defines the set of instances of the range of the role *teaches* which are related, through the role with an instance of the *T* concept (top concept). The equivalence (≡) between *Professor* and $\exists teaches.T$ means that the set of instances of *Professor* is equivalent to the set of instances defined by the existential restriction. According to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model, $\mathcal{S}_0$ can be formalized as below:

$$sC=\{T, Person, Student, Professor\}$$
$$sR=\{teaches\}$$
$$\partial_C=\{(Student, Professor)\}$$
$$\rho_{\exists R}=\{(teaches, T)\}$$
$$\sqcap_C=\{(Person, \rho_{\exists R}(teaches)\}$$
$$\varepsilon_C=\{(Professor, \sqcap_C(Person))\}$$

$$with\ \rho_{\exists R}(teaches)=(teaches, T)\ and\ \sqcap_C(Person)=(Person, \rho_{\exists R}(teaches))$$

Domain experts choose to remove the concept *Professor* by modelling the *deleteConcept* basic change type with *Professor* as parameter. This change is represented during the modelling phase in a log of changes log_0 defined by $log_0 = (deleteConcept,(Professor))$. The application of the *deleteConcept(Professor)* change would turn $\mathcal{S}_0$ structurally inconsistent as other axioms of $\mathcal{S}_0$ uses

Professor as a parameter. According to Table 6, the deletion of a concept C_i requires the deletion of potential axioms of the following axiom subsets:

- sC : The concept C_i itself,
- $\leq_C$: Concept partial orders of C_i,
- σ_R: Roles signatures with C_i in the domain or range,
- σ_A: Attribute signatures with C_i in the domain,
- ι_C: Concept instantiations with C_i as type,
- ε_C: Concept equivalence definitions of C_i,
- δ_C: Concept disjointness definitions of C_i,
- $-_C$: Concept complement definitions of C_i,
- $\sqcap_C$: Concept intersections containing C_i,
- $\sqcup_C$: Concept unions containing C_i,
- $\rho_{\exists R}$: Role universal restrictions with C_i in the range,
- $\rho_{\forall R}$: Role existential restrictions with C_i in the range,

Therefore, to maintain $\mathcal{S}_0$ structural consistency, axioms a_1:={ ∂_C(*Student*)=(*Student, Professor*)} ∈ ∂_C and a_2:={ε_C(*Professor*)=(*Professor,* $\sqcap_C$(*Person*))} ∈ ε_C, wherein *Professor* appears as parameter, must be removed from $\mathcal{S}_0$. The log of changes is then completed with the corresponding deletion changes:

log_0 =((*deleteDisjointConcept, (Student, Professor)*),
(*deleteEquivalentConcept, (Professor,* $\sqcap_C$*(Person)*),
(*deleteConcept(Professor)*))

After the change implementation phase, considering that changes are also logically consistent, the resulting set of axioms of $\mathcal{S}_0$ is defined as below :

sC={T, Person, Student}
sR={teaches}
$\rho_{\exists R}$*={(teaches, T)}*
$\sqcap_C$*={(Person,* $\rho_{\exists R}$*(teaches)}*

with $\rho_{\exists R}$*(teaches)=(teaches, T)*

The corresponding version of $\mathcal{S}_0$ in DL is defined by :

Student
Person ⊓ ∃*teaches.T*

$\mathcal{S}_0$ is assumed structurally consistent as it respects the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model structure.

2.3 Discussion

This section has presented the **structural consistency preventive resolution approach** used in the OntoVersionGraph methodology during the modeling phase of the ontology evolution process. It is based on **structural dependencies** between ontological element deletion changes and axioms taking the deleted element as a parameter. It allows automatically resolving structurally inconsistent changes modelled in the log of changes by adding the required deletion changes. The application example has illustrated the use of this approach on a small ontology from which a concept is deleted. The completed

log of changes has permitted to **handle a structural consistent application of the changes** on the ontology, preserving its structure.

3 Logical Consistency Preventive Approach

This section presents the **logical consistency preventive resolution approach** used to help the user identifying and debugging logically inconsistent sets of changes modelled before applying them on the ontology.

As recommended by (Stojanovic, 2004) the logical consistency preventive resolution intervenes at the **change semantics phase** of the ontology evolution process, after the changes modelled by domain experts have been structurally resolved. It uses the **structural consistent log of changes** produced by the change modelling phase to identify **logically inconsistent minimal sets of changes**. In this perspective, each change operator type definition and logical consequences on the ontology have to be studied in order to assess whether they are compliant to the logical constraints defined by the ontology axioms. When detected, inconsistent changes have to be resolved. As stated in Chapter 4, the resolution approach proposed here is inspired by well known approaches of the **Belief Change** research field, (Flouris & al, 2005). According to (Flouris, 2006) conclusions, two relevant types of belief change approaches are envisioned: **belief revision** and **belief contraction**. These two approaches respectively consist in generating additional changes to the log and removing inconsistent sets of axioms to recover the ontology logical consistency. Belief contraction is a handy approach because the $\mathcal{SHOIN}(\mathcal{D})$ language is **monotonic**. Actually, if the ontology is represented in a monotonic language, **deletion of an axiom does not impact the ontology logical consistency**. Consequently, the proposed approach focuses on **addition changes** impact, i.e. changes adding axioms to the ontology.

The first part introduces the vocabulary related to $\mathcal{SHOIN}(\mathcal{D})$ ontology logical inconsistency used for the formalization of the approach. The second part presents and formalizes the approach considering both basic and composed changes modelled in a log. The third part illustrates the application of the approach on an example of logically inconsistent log of changes.

3.1 $\mathcal{SHOIN}(\mathcal{D})$ Inconsistency

This part gives the definitions inspired by (Baader & al, 2004) of a **model for a $\mathcal{SHOIN}(\mathcal{D})$ ontology**, the **unsatisfiability of an axiom** and the **logical inconsistency of an ontology**, according to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model.

Definition 11 defines a model for a $\mathcal{SHOIN}(\mathcal{D})$ ontology as an interpretation for all its axioms. It means that the $\mathcal{SHOIN}(\mathcal{D})$ ontology has a model if all its axioms can be interpreted on its interpretation domain (c.f. Fig. 16 and Formal Ontology and Interpretation in Chapter 2).

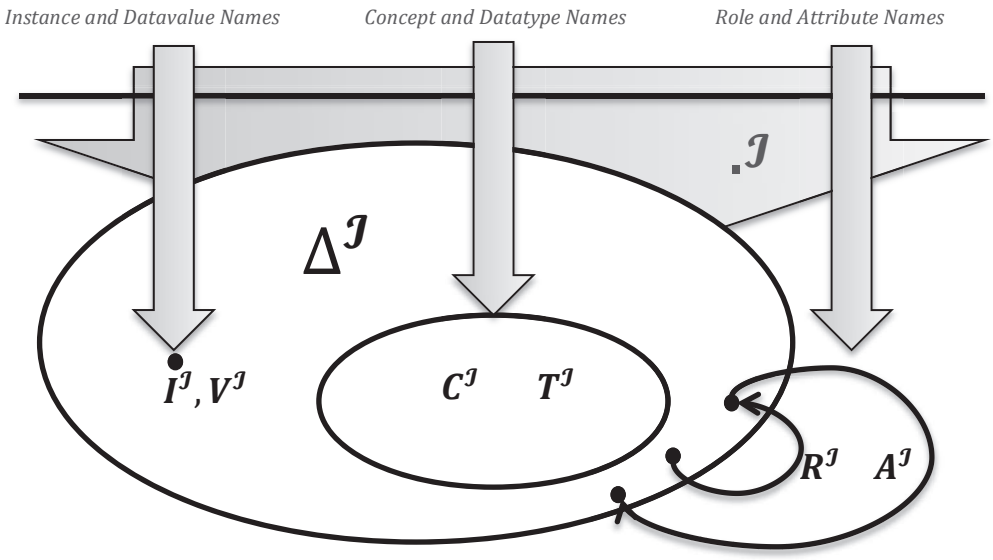


Figure 16. $\mathcal{SHOIN}(\mathcal{D})$ Ontology Simplified Interpretation Domain

Definition 11 : Model of a $\mathcal{SHOIN}(\mathcal{D})$ Ontology.

Let $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)=\{a_1, \ldots, a_n\}$ a $\mathcal{SHOIN}(\mathcal{D})$ ontology, defined as a set of terminological and assertional atomic axioms, where a_i is of the form $a_i :=\{K_i(\varepsilon_1)=(\varepsilon_1, [\varepsilon_2, \ldots, \varepsilon_m])\}$, for each $1 \leq i \leq m$, with $K_i \in \{ \leq_C, \leq_T, \leq_R, \leq_A \sigma_R, \sigma_A, \kappa_R, \kappa_A, \varepsilon_C, \varepsilon_R, \varepsilon_A, \delta_C, \text{-}_C, \text{-}_R, maxCard_R, minCard_R, \sqcap_C, \sqcup_C, \sqcup_I, \sqcup_V, \sqcap_I, \rho_{\exists R}, \rho_{\forall R}, \rho_R, \rho_{\exists A}, \rho_{\forall A}, \rho_A, \iota_C, \iota_T, \iota_R, \iota_A \} \subseteq (\Omega_O, \Sigma_O, \Phi_O)$ an arbitrary $\mathcal{SHOIN}(\mathcal{D})$ construct and $\varepsilon_i \in \{sC, sT, sR, sA, sI, sk_R, sK_A\} \subseteq \Omega_O$ an arbitrary ontological element **;**

Let $\Delta^{\mathcal{I}}$ be a finite set, called the **interpretation domain**.

A mapping $\mathcal{I}$, which interprets $\mathcal{SHOIN}(\mathcal{D})$ elements as subsets of $\Delta^{\mathcal{I}}$**,** is a model of an axiom $a_i :=\{K_i(\varepsilon_1)=(\varepsilon_1, [\varepsilon_2, \ldots, \varepsilon_m])\}$, if, and only if $K_i^{\mathcal{I}}(\varepsilon_i^{\mathcal{I}})=(\varepsilon_i^{\mathcal{I}}, [\varepsilon_i^{\mathcal{I}}, \ldots, \varepsilon_i^{\mathcal{I}}]) \neq \emptyset$. A model for an ontology $\mathcal{S}_O$ is an **interpretation**, which is **a model for all axioms** in $\mathcal{S}_O$.

Definition 12: Unsatisfiability of an Axiom. An axiom a_i is **unsatisfiable** w.r.t. a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O$ if, and only if, $a_i^{\mathcal{I}} = \emptyset$ for all models $\mathcal{I}$ of $\mathcal{S}_O$.

Based on this semantics, a $\mathcal{SHOIN}(\mathcal{D})$ ontology can be checked for **inconsistency**, i.e., whether there are unsatisfiable axioms: axioms, which are necessarily interpreted as the empty set in all models of $\mathcal{S}_O$.

Definition 13: Logical Inconsistency of a $\mathcal{SHOIN}(\mathcal{D})$ Ontology. A $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$ is **logically inconsistent** if there exists a set of axioms in $\mathcal{S}_O$ which is unsatisfiable, i.e. $\mathcal{S}_O$ has no models.

According to these definitions, checking logical consistency of an ontology amounts to detect the existence of any unsatisfiable set of axioms among the axioms of the ontology. An **atomic $\mathcal{SHOIN}(\mathcal{D})$ axiom** $a_i :=\{K_i(\varepsilon_1)=(\varepsilon_1, [\varepsilon_2, \ldots, \varepsilon_m])\}$ is described by a $\mathcal{SHOIN}(\mathcal{D})$ construct K_i taking ontological elements ε_i as parameters, which is logically defined by constraints. The **satisfiability** of such axiom depends on the satisfaction, by its parameters, of all the **logical constraints of the whole set of axioms** in the ontology. So for each addition of an axiom to an ontology, the set of logical constraints defined by the whole set of existing axioms has to be satisfied. This logical verification is not required for the deletion of an axiom. Indeed the deletion of an axiom from a $\mathcal{SHOIN}(\mathcal{D})$ ontology logically amounts to remove its corresponding constraints from the whole set of logical constraints. Consequently, during the ontology

evolution semantic phase, **checking logical consistency amounts to check the satisfaction of the whole set of logical constraints w.r.t changes adding axioms**.

3.2 $\mathcal{SHOIN}(\mathcal{D})$ Logical Inconsistency Preventive Approach

This part formalizes the **logical inconsistency preventive approach** proposed in the OntoVersionGraph methodology. It explains how to derive **unsatisfiability constraints** from basic and composed change operator types definitions and describes the **logical inconsistency identification process** to detect minimal inconsistent sets of changes in the log of changes according to unsatisfiability constraints.

3.2.1 Basic $\mathcal{SHOIN}(\mathcal{D})$ Change Unsatisfiability Constraints

Basic changes corresponding to additions of axioms are called **basic addition changes** as defined in Chapter 4 by the $\mathcal{SHOIN}(\mathcal{D})$ ontology model. 42 basic addition change operator types have been structurally defined in this model. Among them, only **33 require a logical definition of their construct constraints**. The six change operator types adding ontological elements, such as concept, datatype, role, attribute, instance and datavalue (respectively *AddConcept, AddDatatype, AddRole, AddAttribute, AddInstance* and *AddDatavalue* change operator types), are not logically constrained. Also the three changes adding unions of existing concepts, enumerations of existing instances and enumerations of existing datavalues (respectively *AddConceptUnion, AddInstanceEnumeration* and *AddDatavalueEnumeration* change operator types), are not either constrained. They just add elements and simple sets of elements to the model, which logical satisfiability does not depend on the existing axiom constraints. **Unsatisfiability constraints** have been defined for each of the 33 constrained basic addition changes, according to the $\mathcal{SHOIN}(\mathcal{D})$ language model. To define the unsatisfiability constraints of a basic addition change, the **change definition** and **its logical consequences** are firstly studied.

For example, the *AddEquivalentConcept* change operator type is defined by the addition of an equivalence constraint between two concepts *D1* and *D2*. Structurally, it adds an axiom $a_i:=\{\varepsilon_C(D1)_i=(D1, D2)\}$, with $i \in \mathbb{N}$, to the equivalence subset ε_C of a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$. Logically, it implies several consequences as illustrated in Figure 15 below:

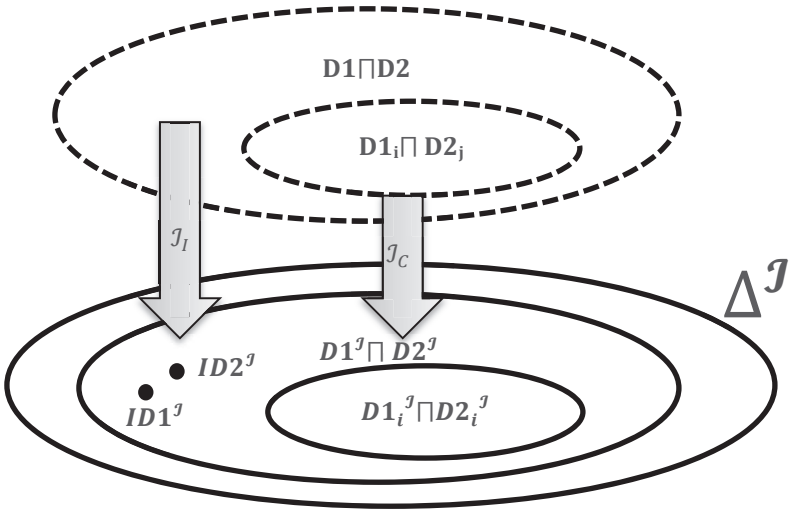


Figure 17. Interpretation of an Equivalence Constraint on Two Concepts D1 and D2

According to Fig. 17, the logical consequences are:

- *D1* and *D2* are equivalent to their intersection.

- *D1* and *D2* share the same sub-concepts.
- *D1* and *D2* have the same interpretation and therefore have the same instances.

The corresponding unsatisfiability constraints and their **impact on each axioms subset of the $\mathcal{SHOIN}(\mathcal{D})$ ontology** can secondly be evaluated. The logical consequences of the equivalence of *D1* and *D2* imply the following unsatisfiability constraints:

– The intersection of *D1* and the complement of *D2*, and the intersection of *D2* with the complement of *D1*, are unsatisfiable.
– The disjunction of *D1* and *D2* is unsatisfiable.
– The definition of *D1* and *D2* as subconcepts of the complement of *D1* or *D2* is unsatisfiable.
– The intersection of one of *D1* and *D2* sub-concepts with the complement of *D1* or *D2*, or one of its subconcepts is unsatisfiable.
– The instantiation of *D1* or *D2* defined as an instance of the complement of *D1* or *D2* is unsatisfiable.

This list is not exhaustive but can be completed considering unsatisfiability constraints **also applied on D1 and D2 sub-concepts, equivalent concepts and intersected concepts.**

For each change operator type, unsatisfiability constraints are formalized as **generic sets of axioms**. These sets, if existing in the ontology, **cause an inconsistency if the change is performed**. For example, the application of a *AddEquivalentConcept* change with two concepts *D1* and *D2* as parameters turns the ontology inconsistent if the **concept** disjunction axiom a_i:={$\delta_C(D1)_i$=(D1,D2)} corresponding to the second unsatisfiability constraint above, is already present in the ontology. Table 7 shows the whole set of unsatisfiability constraints of the *AddEquivalentConcept* change in terms of inconsistent generic sets of axioms. The first column shows the list of parameters, from the $\mathcal{SHOIN}(\mathcal{D})$ ontological element types, required to represent a change. The second column gives the axiom content added by such change. The third column lists the corresponding generic inconsistent sets of axioms to check in the log of changes before applying the change.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Concepts *D1, D2*	$\varepsilon_C(D1)_i$=(D1,D2)	• *if* $\leq_C (D2) = \{(D2, D2_1), \dots, (D2, D2_n)\}$ *and* $\exists i\ -_C(D1)_i$=(D1,D1-) *and* $\exists j$ *such that* $\leq_C(D1\text{-})_j$=(D1-,$D2_j$) • *if* $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$ *and* $\exists i\ -_C(D2)_i$=(D2,D2-) *and* $\exists j$ *such that* $\leq_C(D2\text{-})_j$=(D2-,$D1_j$) • *if* $\exists i\ -_C(D1)_i$=(D1,D1-) *and* $\exists j\ \varepsilon_C(D2)_j$=(D2,D1-) • *if* $\exists i\ -_C(D2)_i$=(D2,D2-) *and* $\exists j\ \varepsilon_C(D1)_j$=(D1,D2-) • *if* $\exists i\ \delta_C(D1)_i$=(D1,D2) • *if* $\exists i\ \delta_C(D2)_i$=(D2,D1) • *if* $\exists i\ -_C(D1)_i$ =(D1,D2) • *if* $\exists i\ -_C(D2)_i$ =(D2,D1) • *if* $\iota_C(D1)$={(D1,$I1_1$),…,(D1,$I1_n$)} *and* $\exists i\ -_C(D1)_i$=(D1,D1-) *and* $\exists j$ *such that* $\iota_C(D1\text{-})_j$=(D1-, $I2_j$) • *if* $\iota_C(D2)$={(D2,$I2_1$),…,(D2,$I2_n$)} *and* $\exists i\ -_C(D2)_i$=(D2,D2-) *and* $\exists j$ *such that* $\iota_C(D2\text{-})_j$=(D2-, $I1_j$) • *if* $\exists i\ -_C(D1)_i$ =(D1,D1-) *and* $\iota_C(D1\text{-})$={(D1-,$I1\text{-}_1$),…,(D1-,$I1\text{-}_n$)} *and* $\exists j$ *such that* $\varepsilon_I(I1\text{-}_j)_j = (I1\text{-}_j, I2_i)$ • *if* $\exists i\ -_C(D2)_i$ =(D2,D2-) *and* $\iota_C(D2\text{-})$={(D2-,$I2\text{-}_1$),…,(D2,$I2\text{-}_n$)} *and* $\exists j$ *such that* $\varepsilon_I(I2\text{-}_j) = (I2\text{-}_j, I1_i)$

Table 7. Unsatisfiability Constraints of the AddEquivalentConcept Change Operator Type Formalized as Generic Inconsistent Axioms Sets.

A non-exhaustive list of unsatisfiability constraints for the other 32 basic addition change operator types is given in Annex 1. This list is non-exhaustive as it is still a work in progress. The following paragraph formalizes the approach studying the definition of potential composed change operator types, built upon the composition of several basic change operator types, in order to define their corresponding unsatisfiability constraints

3.2.2 Composed $\mathcal{SHOIN}(\mathcal{D})$ Change Unsatisfiability Constraints

Composed changes are defined as **changes aggregating basic ones**. They can be composed by **both basic addition** and **deletion changes** (e.g. *renameConcept* example in Chapter 4). To evaluate their impact on an ontology logical consistency, it is therefore essential to decompose them into their corresponding basic changes. The unsatisfiability constraints of a compose change operator type can then be built by **composing the unsatisfiability constraints of the set of basic addition changes** included in their composition.

Let us focus on a commonly used composed change, which defines a concept *D* as the intersection of a concept *D1* and the existential restriction of a role *R* to a range concept *D2*. This composed change could be called *AddEquivalentConceptIntersectionExistentialRestriction*. The non-atomic axiom added by this change can be modelled as below in DL: $D \equiv D1 \sqcap \exists R.D2$. This generic axiom could, for instance, describe a mother as a woman having at least one child, which is a human: a concept *Mother* is therefore defined as the intersection of the concept *Woman* with a role existential restriction on a role *hasChild* and a concept *Human*. In DL : $Mother \equiv Woman \sqcap \exists hasChild.Human.$ The generic non-atomic axiom is composed of three atomic axioms, depending on each other, which can be defined as below according to the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model :

$$a_1 := \{\varepsilon_C(D) = (D, a_2)\}$$
$$a_2 := \{\sqcap_C(D1) = (D1, a_3)\}$$
$$a_3 := \{\rho_{\exists R}(R) = (R, D2)\}$$

a_3 describes a role existential restriction on a role R and a concept $D2$. a_3 defines a non-atomic concept, which is not named. a_2 describes a concept intersection between a concept $D1$ and the concept defined by the axiom a_3. a_2 also defines an anonymous non-atomic concept. a_1 describes a concept equivalence between a concept D and the concept defined by the axiom a_2. The basic addition change operator types representing the addition of a_1 , a_2 and a_3 respectively are the *addRoleExistentialRestriction,* the *addConceptIntersection* and the *addConceptEquivalence* change operator types. Unsatisfiability constraints have been defined in Annex 1 for each of these basic addition changes. Therefore it is easy to derive the unsatisfiability constraints of their composition:

- First, replacing the second parameter of the role existential restriction addition change by the generic concept *D2*, allows defining the first set of unsatisfiability constraints as formalized in Table 8a below.

Entities	**$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom**	**Unsatisfiability Constraints**
Role *R*, Concept *D2*	$\rho_{\exists R}(R)_i=(R,D2)$	• $If \leq_{Rsub} (R) = \{(R, R_1), \dots, (R, R_n)\}$ *and* $\exists i\ \rho_{\exists R}(R_i) = \{(R_i, D_{i1}), \dots, (R_i, D_{in})\}$ *such that* $\forall j\ D_{ij} \not\equiv D2$ • $If \exists i\ \varepsilon_R(R)_i = Re$ *and* $\rho_{\exists R}(Re) = \{Re, De_1), \dots, (Re, De_n)\}$ *such that* $\forall j\ D2 \neq De_j$ • $if\ \iota_R(R) = \{(R,(iD_1,iRa_1)),\dots,(R,(iD_n,iRa_1))\}$ *and* $\exists i$ *such that* $\forall j\ iRa_i \notin D2$

Table 8a. $\mathcal{SHOIN}(\mathcal{D})$ Composed Change Example Unsatisfiability Constraints part 1

- Second, replacing both parameters of the concept intersection addition change by the generic concept *D1* and the anonymous concept defined by the axiom a_3 allows defining the second set of unsatisfiability constraints.

Concepts $D1, a_3$	$\sqcap_C(D1)=(D1, a_3\})$	• *If* $\exists i \leq_C (a_3) = \{(a_3, a_{3_1}), \dots, (a_3, a_{3_n})\}$ *and If* $\exists j -_C (D1)_j=(D1,D1\text{-})$ *and* $\exists k$ *such that* $\varepsilon_C(a_3)_k= (a_3,D1\text{-})$ • *If* $\exists i \leq_C (a_3) = \{(a_3, a_{3_1}), \dots, (a_3, a_{3_n})\}$*and* $\exists j$ *such that* $\delta_C(a_3)_j=(a_3, D1)$ • *If* $\exists i -_C (D1)_i=(D1,D1\text{-})$ *and* $\exists j\ \iota_C(a_3)=\{(a_3,Ia_{3\,1}),\dots,(a_3,Ia_{3\,n})\}$ *and* $\exists k\ \iota_C(D1\text{-})_k= (D1\text{-},Ia_{3\,k})$ • *If* $\exists i -_C (D1)_i=(D1,D1\text{-})$ *and* $\exists j\ \exists k\ \iota_C(a_3)_k=(a_3,Ia_{3\,k})$ *and* $\exists l\ \iota_C(D1\text{-})_l=(D1\text{-},I\text{-}_l)$ *such that* $\varepsilon_I(Ia_{3\,k}) = (Ia_{3\,k}, I\text{-}_l)$ • *If* $\exists i -_C (D1)_i=(D1,D1\text{-})$ *and* $\exists j\ \exists k\ \iota_C(a_3)_k=(a_3,Ia_{3\,k})$ *and* $\exists l\ \iota_C(D1\text{-})_l=(D1\text{-},I\text{-}_l)$ *such that* $\varepsilon_I(I\text{-}_l) = (I\text{-}_l, Ia_{3\,k})$ • *If* $\exists i -_C (D1)_i=(D1,D1\text{-})$ *and* $\exists j\ \varepsilon_C(a_3)=(a_3, D1\text{-})$ • *If* $\exists i\ \delta_C(D1)=(D1, a_3)$ • *If* $\exists i\ \delta_C(a_3)_i=(a_3, D1)$ • *If* $\exists i -_C (D1)_i=(D11\text{-})$ *and* $\exists j\ \leq_C (a_3) = (a_3, D1\text{-})$ • *If* $\exists i -_C (D1)_i=(D1,D1\text{-})$ *and* $\leq_C(D1\text{-})=(D1\text{-}, Y)$

Table 8b. $\mathcal{SHOIN}(\mathcal{D})$ Composed Change Example Unsatisfiability Constraints part 2

- Third, replacing both parameters of the concept equivalence addition change by the generic concept D and the anonymous concept defined by the axiom a_2 allows defining the third set of unsatisfiability constraints.

Concepts D, a_2	$\varepsilon_C(D)_i=(D, a_2)$	• *if* $\leq_C (a_2) = \{(a_2, a_{2_1}), \dots, (a_2, a_{2_n})\}$ *and* $\exists i -_C(D)_i=(D,D\text{-})$ *and* $\exists j$ *such that* $\leq_C(D\text{-})_j=(D\text{-},a_{2\,j})$ • *if* $\leq_C (D) = \{(D, D_1), \dots, (D, D_n)\}$ *and* $\exists i -_C(a_2)_i=(a_2, a_2\text{-})$ *and* $\exists j$ *such that* $\leq_C(a_2\text{-})_j=(a_2\text{-},D_j)$ • *if* $\exists i -_C(D)_i=(D,D\text{-})$ *and* $\exists j\ \varepsilon_C(a_2)_j=(a_2,D\text{-})$ • *if* $\exists i -_C(a_2)_i=(a_2, a_2\text{-})$ *and* $\exists j\ \varepsilon_C(D)_j=(D, a_2\text{-})$ • *if* $\exists i\ \delta_C(D)_i=(D, a_2)$ • *if* $\exists i\ \delta_C(a_2)_i=(a_2,D)$ • *if* $\exists i -_C(D)_i=(D, a_2)$ • *if* $\exists i -_C(a_2)_i=(a_2,D)$ • *if* $\iota_C(D)=\{(D,I_1),\dots,(D,I_n)\}$ *and* $\exists i -_C(D)_i=(D,D\text{-})$ *and* $\exists j$ *such that* $\iota_C(D\text{-})_j=(D\text{-}, a_{2\,j})$ • *if* $\iota_C(a_2)=\{(a_2,Ia_{2\,1}),\dots,(a_2,Ia_{2\,n})\}$ *and* $\exists i -_C(a_2)_i=(a_2, a_2\text{-})$ *and* $\exists j$ *such that* $\iota_C(a_2\text{-})_j=(a_2\text{-}, I_j)$ • *if* $\exists i -_C(D)_i=(D,D\text{-})$ *and* $\iota_C(D\text{-})=\{(D\text{-},I\text{-}_1),\dots,(D\text{-},I\text{-}_n)\}$ *and* $\exists j$ *such that* $\varepsilon_I(I\text{-}_j)_j = (I\text{-}_j, Ia_{2\,i})$ • *if* $\exists i -_C(a_2)_i=(a_2, a_2\text{-})$ *and* $\iota_C(a_2\text{-})=\{(a_2\text{-},Ia_{2\,\text{-}1}),\dots,(a_2,Ia_{2\,\text{-}n})\}$ *and* $\exists j$ *such that* $\varepsilon_I(Ia_{2\,\text{-}j}) = (Ia_{2\,\text{-}j}, I_j)$

Table 8c. Composed Change Example Unsatisfiability Constraints part 3

3.2.3 Logical Consistency Preventive Approach Process

Changes logical consistency is assessed during the change semantic phase of the ontology evolution process, after the change modelling phase. The preventive approach process proposed in OntoVersionGraph is composed of two steps: **logical consistency detection** and **logical consistency resolution**. The **first is automated** whereas the **second requires the domain experts' intervention**.

The logical consistency detection process uses the structurally consistent log of changes log_i produced by the change modelling phase. First, log_i is concatenated to a copy of the ontology global log of changes log_0 such that all the changes added to the ontology and the set of new modelled ones are temporary listed in the same log log_{temp}. Second, log_{temp} is developed in order to decompose every change it contains as basic ones. Composed changes are therefore developed into the basic ones composing them. Third, for each addition change decomposed in log_{temp}, the corresponding unsatisfiability constraints are extracted and listed in a list $listOfUC_i$. As stated above, unsatisfiability constraints are defined for each change in terms of **minimal generic sets of axioms** turning an ontology inconsistent if they are already present in the ontology and if the change is applied. Therefore, $listOfUC_i$ contains the whole set of minimal generic inconsistent sets of axioms. Fourth, the **generic inconsistent sets of axiom parameters are then instantiated with the ontological elements** represented as change parameters in log_{temp}, according

to each change logical constraints. It allows generating the **potential minimal inconsistent sets of changes**, still represented in $listOfUC_i$, which have to be checked in log_{temp}. Fifth, if one of the minimal inconsistent sets of changes of represented in $listOfIUC_i$ is found in log_{temp}, it means that the change from which this set has been generated, will cause an inconsistency if applied. Sixth, the detected minimal inconsistent sets of changes are returned to the domain experts, which have to choose and agree on a **resolution approach**.

Once, the detected minimal inconsistent sets of changes are known by the domain experts, **two resolution approaches** inspired by belief revision and belief contraction techniques are firstly proposed. According to **belief revision**, they can choose to generate additional changes to the log of changes log_i to resolve the inconsistencies (Ribeiro & Wasserman, 2009b). In this case, a new change modeling phase is undertaken to check the structural consistency of the resolution changes modelled and, if successfully passed, it will be followed by a new change semantics phase. Otherwise, according to **belief contraction** (Ribeiro & al, 2009), they can remove one or more changes according to each detected minimal inconsistent set of changes from the log of changes log_i. In this case no more change modelling phase is required, the change semantics phase can be reloaded to directly assess the logical consistency of the updated log_i. Finally, the changes represented in log_i are assessed as logically consistent according to the unsatisfiability constraints provided by OntoVersionGraph if no more minimal inconsistent set of $listOfIUC_i$ is detected in log_{temp}.

3.3 Logical Consistency Preventive Analysis Example

This part illustrates the logical consistency preventive approach process with an example of logical inconsistency detection and resolution. Let us consider the following consistent set of axioms defining an ontology $\mathcal{S}_0$, where *Person*, *Student* and *Professor* are defined concept names:

$$Student \sqcap Professor \sqsubseteq \bot$$
$$Professor \equiv Person \sqcap \exists teaches.\top$$
$$Student \equiv Person \sqcap \exists attends.\top$$

Concepts *Student* and *Professor* are defined as disjoint concepts. The role existential restriction $\exists teaches.T$ defines the set of instances of the range of the role *teaches* which are related, through the role with an instance of the T concept (*TopConcept*). The concept equivalence ($\equiv$) between *Professor* and $\exists teaches.T$ means that the set of instances of *Professor* is equivalent to the set of instances defined by the existential restriction $\exists teaches.T$. It means that a professor can teach something (like a course). We have similar semantics for the concept *Student* and the role existential restriction $\exists attends.T$. A student can attend something. The concept disjunction constraint ($\doteq$) between *Student* and *Professor* semantically implies that it is impossible for an instance to teach and attends something if it is a member of the set of instances of *Student* or *Professor*.

Let us add a new concept, named *PhDStudent*, which is defined as a subconcept of the concept intersection between *Student* and the concept defined by the role existential restriction $\exists\ teaches.T$. The definition of *PhDStudent* is added to $\mathcal{S}_0$ as below:

$$Student \sqcap Professor \sqsubseteq \bot$$
$$Professor \equiv Person \sqcap \exists teaches.\top$$
$$Student \equiv Person \sqcap \exists attends.\top$$
$$PhDStudent \sqsubseteq Student \sqcap \exists teaches.\top$$

The definition of the *PhDStudent* concept contradicts the concept disjunction constraint between *Student* and *Professor*. As a professor is defined as a person who can teach, then a student is automatically defined as a person who cannot teach. The concept intersection $Student \sqcap \exists\ teaches.T$ is then

unsatisfiable, as there is no instance in the model of $\mathcal{S}_0$, which could be typed with the concept defined by this intersection. $\mathcal{S}_0$, is then inconsistent.

This logical inconsistency could have been identified and resolved before the application of the changes on the ontology $\mathcal{S}_0$, by applying the logical consistency preventive approach proposed in OntoVersionGraph.

3.3.1 Logical Consistency Detection Example

According to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model axioms of the first version of $\mathcal{S}_0$ can be defined by:

$$a_0 := \{sC(Person)\}$$
$$a_1 := \{sC(Professor)\}$$
$$a_2 := \{sC(Student)\}$$
$$a_3 := \{\partial_C(Professor, Student)\}$$
$$a_4 := \{sR(teaches)\}$$
$$a_5 := \{sR(attends)\}$$
$$a_6 := \{\rho_{\exists R}(teaches, TopConcept)\}$$
$$a_7 := \{\rho_{\exists R}(attends, TopConcept)\}$$
$$a_8 := \{\sqcap_C(Person, a_6)\}$$
$$a_9 := \{\sqcap_C(Person, a_7)\}$$
$$a_{10} := \{\varepsilon_C(a_8, Professor)\}$$
$$a_{11} := \{\varepsilon_C(a_9, Student)\}$$

$\mathcal{S}_0$ can then described by :

$sC=\{TopConcept, Person, Professor, Student, a_6, a_7, a_8, a_9\}$,
$sR=\{teaches, attends\}$,
$\varepsilon_C=\{(a_8, Professor), (a_9, Student)\}$,
$\partial_C=\{(Professor, Student)\}$,
$\sqcap_C=\{(Person, a_6), (Person, a_7)\}$,
$\rho_{\exists R}=\{(teaches, TopConcept), (attends, TopConcept)\}$.

The global log of changes log_0 *of* $\mathcal{S}_0$, recording all the changes applied on $\mathcal{S}_0$ since its creation is assumed defined by :

$log_0=(\beta_{B_0}, \beta_{B_1}, \beta_{B_2}, \beta_{B_3}, \beta_{B_4}, \beta_{B_5}, \beta_{B_6}, \beta_{B_7}, \beta_{B_8}, \beta_{B_9}, \beta_{B_{10}}, \beta_{B_{11}})$ with:

β_{B_0} = addConcept(Person),
β_{B_1} = addConcept(Professor),
β_{B_2} =addConcept(Student),
β_{B_3} =addDisjointConcept(Professor, Student),
β_{B_4} =addRole(teaches),
β_{B_5} =addRole(attends),
β_{B_6} =addRoleExistentialRestriction(teaches, T),
β_{B_7} =addRoleExistentialRestriction(attends, T),
β_{B_8} =addConceptIntersection(Person, a_6),
β_{B_9} =addConceptIntersection(Person, a_7),
$\beta_{B_{10}}$ =addEquivalentConcept(a_8, Professor),
$\beta_{B_{11}}$ =addEquivalentConcept(a_9, Student).

The logical consistency detection handled in the change semantics phase uses the log *log1* representing the new changes to apply on $\mathcal{S}_0$ by domain experts. It is assumed structurally consistent according to the change modeling phase of the corresponding ontology evolution process from which it is issued. *log1* defined by:

$log_1=(addConcept(PhDStudent), addConceptIntersection(Student, a_6), addSubConcept(a_{13}, PhDStudent))$

Also changes represented in log_1 intend to add to $\mathcal{S}_0$ the following axioms:

a_{12} :={sC(PhDStudent)}
a_{13} :={$\sqcap_C$(Student, a_6)}
a_{14} :={$\leq_C$(a_{13}, PhDStudent)}

According to the logical consistency detection process, log_1 is firstly concatenated to log_0 to represent in the same log the changes applied on $\mathcal{S}_0$ since its creation and the new changes. The log produced from this concatenation, noted log_{temp}, is defined by:

log_{temp}=(β_{B_0}, β_{B_1}, β_{B_2}, β_{B_3}, β_{B_4}, β_{B_5}, β_{B_6}, β_{B_7}, β_{B_8}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, $\beta_{B_{12}}$, $\beta_{B_{13}}$, $\beta_{B_{14}}$) with:

β_{B_0} = addConcept(Person),
β_{B_1} = addConcept(Professor),
β_{B_2} =addConcept(Student),
β_{B_3} =addDisjointConcept(Professor, Student),
β_{B_4} =addRole(teaches),
β_{B_5} =addRole(attends),
β_{B_6} =addRoleExistentialRestriction(teaches, T),
β_{B_7} =addRoleExistentialRestriction(attends, T),
β_{B_8} =addConceptIntersection(Person, a_6),
β_{B_9} =addConceptIntersection(Person, a_7),
$\beta_{B_{10}}$ =addEquivalentConcept(a_8, Professor),
$\beta_{B_{11}}$ =addEquivalentConcept(a_9, Student),
$\beta_{B_{12}}$ =addConcept(PhDStudent),
$\beta_{B_{13}}$ =addConceptIntersection(Student, a_6),
$\beta_{B_{14}}$ =addSubConcept(a_{13}, PhDStudent).

Then each addition change of log_{temp} is separately analysed to extract the satisfiability constraints corresponding to its change operator type (see column CTUCE on Table 9 below). Unsatisfiability constraints are instantiated on the ontological elements change represented as parameters of log_{temp} changes (see column CTUCI on Table 9 below). Finally, the potential minimal inconsistent sets of changes are generated (see column CIA on Table 9 below) from the unsatisfiability constraints for which all parameters have been instantiated (in bold in column CTUCI on Table 9 below).

Change Operator Types Unsatisfiability Constraint Extraction (CTUCE)		Change Operator Types Unsatisfiability Constraint Instantiation (CTUCI)		Change Inconsistency Analysis (CIA)
log_{temp} Changes Chronology	**Change Operator Types Unsatisfiability Constraints**	**Ontological Elements added by Changes**	**Potential Minimal Inconsistent Sets of Axioms added by log_{temp} Changes**	**Potential Inconsistent Set of Changes To Check in log_1**
β_{B_0}=*addConcept(Person)*	addConcept(D)	*Person* $\in sC$		
β_{B_1}=*addConcept(Professor)*		*Professor*$\in sC$		
β_{B_2}=*addConcept(Student)*		*Student* $\in sC$		
β_{B_3}=*addDisjointConcept(Professor, Student)*	addDisjointConcept(D1,D2) • if $\leq_C$ (D1, $D1_1$), … , (D1, $D1_n$) and $\leq_C$ ($D1$, D2) • if $\leq_C$ (D2, $D2_1$), … , (D2, $D2_n$) and $\leq_C$ ($D2$, D1) • if ε_C(D1,D2) • if ε_C(D2,D1) • if $\sqcap_C$(D1,D2) • if $\sqcap_C$(D2,D1) • if $-_C$(D2,D2-) and δ_C(D1, D2-) • if $-_C$(D1,D1-) and δ_C(D2, D1-) • if ι_C(D2,$I2_1$),…,(D2,$I2_n$) and ∃i such that ι_C($D1$,$I2_i$) • if ι_C(D1,$I1_1$),…,(D1,$I1_n$) and ∃i such that ι_C($D2$,$I1_i$) • if ∃i, j ι_C(D2,$I2_1$),…,(D2,$I2_n$) and ι_C(D1,$I1_1$),…,(D1,$I1_n$)such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(D2,$I2_1$),…,(D2,$I2_n$) and ι_C(D1,$I1_1$),…,(D1,$I1_n$) such that ε_I($I2_j$, $I1_i$)		β_{B_3}→ • if $\leq_C$((Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$(Student, $D2_1$), … , (Student, $D2_n$)) and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(Student, Professor)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Professor, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C Professor,$I1_1$),…,(Professor, $I1_n$)} and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$)	***β_{B_3}, addEquivalentConcept(Professor, Student)*** ***β_{B_3}, addEquivalentConcept(Student, Professor)*** ***β_{B_3}, addConceptIntersection(Professor, Student)*** ***β_{B_3}, addConceptIntersection(Student, Professor)***
β_{B_4}=*addRole(teaches)*	addRole(R)	*teaches* $\in sR$	β_{B_3}→ • if $\leq_C$ (Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$(Student, $D2_1$), … , (Student, $D2_n$) and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(Student, Professor)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Professor, Student)** • if $-_C$(Student)$_i$=(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Professor, D1-) and δ_C(Student)$_j$=(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$)} and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I$$I1_i$, $I2_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$)	*β_{B_3}, addEquivalentConcept(Professor, Student)* *β_{B_3}, addEquivalentConcept(Student, Professor)* *β_{B_3}, addConceptIntersection(Professor, Student)* *β_{B_3}, addConceptIntersection(Student, Professor)*
β_{B_5}=*addRole(attends)*		*attends* $\in sR$	β_{B_3}→ • if $\leq_C$ (Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$(Student, $D2_1$), … , (Student, $D2_n$) and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(Student, Professor)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Professor, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I1_i$, $I2_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$)} such that ε_I($I2_j$, $I1_i$)	*β_{B_3}, addEquivalentConcept(Professor, Student)* *β_{B_3}, addEquivalentConcept(Student, Professor)* *β_{B_3}, addConceptIntersection(Professor, Student)* *β_{B_3}, addConceptIntersection(Student, Professor)*
β_{B_6}=*addRoleExistentialRestriction(teaches, TopConcept)*	addRoleExistentialRestriction(R, D) • if $\leq_R$ (R) = {(R, R_1), … , (R, R_n)} and ∃i $\rho_{\exists R}$(R_i) = {(R_i, D_1), … , (R_i, D_n)} such that ∀j $D_i \not\sqsubseteq D$ • If ∃i ε_R(R)$_i$=(R,Re) and $\rho_{\exists R}(Re)$ = {Re, De_1), … , (Re, De_n)} such that ∀j D ≠ De_j • if $\iota_R(R)$={(R,(iD_1,iRa_1)),…,(R,(iD_n,iRa_1))} and	*a_6 :={ $\rho_{\exists R}$ (teaches)=(teaches, TopConcept)}* $\in sC$	β_{B_3}→ • if $\leq_C$ (Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)) and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(Student, Professor)** • **if $\sqcap_C$(Student, Professor)**	*β_{B_3}, addEquivalentConcept(Professor, Student)* *β_{B_3}, addEquivalentConcept(Student, Professor)* *β_{B_3}, addConceptIntersection(Professor, Student)* *β_{B_3}, addConceptIntersection(Student, Professor)*

	∃i such that ∀j $iRa_i \notin$		• **if $\sqcap_C$(Professor, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$)} and ∃i such that ι_CProfessor, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I(I1_i, I2_j)$ • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I(I2_j, I1_i)$ β_{B_6}→ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n and ∃i $\rho_{\exists R}(R_i, D_1)$, … , (R_i, D_n) such that ∀j $D_i \notin T$ • If ∃i ε_R(teaches,Re) and $\rho_{\exists R}$(Re, De_1), … , (Re, De_n) such that ∀j T ≠ De_j • if ι_R(teaches,(iD_1,iRa_1)),…,(teaches,(iD_n,iRa_1)) and ∃i such that ∀j $iRa_i \notin T$	
β_{B_7} =addRoleExistentialRestriction(attends, TopConcept)		a_7 :={ $\rho_{\exists R}$ (attends)={attends, TopConcept)} ∈ sC	β_{B_3}→ • if $\leq_C$ (Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$) and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(Student, Professor)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Professor, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I(I1_i, I2_j)$ • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$)} and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$)} such that $\varepsilon_I(I2_j, I1_i)$ β_{B_6}→ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n)) and ∃i $\rho_{\exists R}(R_i, D_1)$, … , (R_i, D_n) such that ∀j $D_i \notin T$ • If ∃i ε_R(teaches,Re) and $\rho_{\exists R}$(Re, De_1), … , (Re, De_n) such that ∀j T ≠ De_j • if ι_R (teaches,(iD_1,iRa_1)),…,(teaches,(iD_n,iRa_1)) and ∃i such that ∀j $iRa_i \notin T$ β_{B_7}→ • If $\leq_R$ (attends, R_1), … , (attends, R_n) and ∃i $\rho_{\forall R}(R_i, D_1)$, … , (R_i, D_n)) such that ∀j $D_i \notin T$ • If ∃i ε_R(attends,Re) and $\rho_{\forall R}$(Re, De_1), … , (Re, De_n)) such that ∀j T ≠ De_j • if ι_R(attends,(iD_1,iRa_1)),…,(attends,(iD_n,iRa_1)) and ∃i such that ∀j $iRa_i \notin T$	β_{B_3}, addEquivalentConcept(Professor, Student) β_{B_3}, addEquivalentConcept(Student, Professor) β_{B_3}, addConceptIntersection(Professor, Student) β_{B_3}, addConceptIntersection(Student, Professor)
β_{B_8} =addConceptIntersection(Person, a_6)	addConceptIntersection(D1, D2) • If ∃i (Di, Di_1), … , (Di, Di_n) and If ∃j $-_C$(Dij,Djj) • If ∃i $\leq_C$ (Di, Di_1), … , (Di, Di_n) and ∃j∃k∃l such that δ_C(Dij,Dkl) • If ∃i ∃j$-_C$(Di,Dj) • If ∃i $-_C$(Di,Di-) and ∃j ι_C(Dj,Ij_1),…,(Dj,Ij_n) and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Di,Di-) and ∃j ∃k ι_C(Dj,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(Di,Di-) and ∃j ∃k ι_C(Dj,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l} ,Ij_k) • If ∃i $-_C$(Di,Di-) and ∃j ε_C(Dj, Di-) • If ∃i ∃j δ_C(Di, Dj)	a_8 :={$\sqcap_C$(Person)={Person, a_6)} ∈ sC	β_{B_3}→ • if $\leq_C$((Professor, $D1_1$), … , (Professor, $D1_n$)and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and ∃i such that $\leq_C$ (a_8, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(a_8, Student)** • **if ε_C(Student, Professor)** • **if ε_C(Student, a_8)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Student, a_8)** • **if $\sqcap_C$(Professor, Student)** • **if $\sqcap_C$(a_8, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Student, D2-) and δ_C(a_8, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if $-_C$(a_8, D1-) and δ_C(Student, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(a_8, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ι_C(a_8,$I1_1$),…,(a_8, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I(I1_i, I2_j)$ • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that $\varepsilon_I(I1_i, I2_j)$ • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that $\varepsilon_I(I2_j, I1_i)$ • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$)} and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that $\varepsilon_I(I2_j, I1_i)$	β_{B_3}, addEquivalentConcept(Professor, Student) β_{B_3}, addEquivalentConcept(Student, Professor) β_{B_3}, addConceptIntersection(Professor, Student) β_{B_3}, addConceptIntersection(Student, Professor) **β_{B_8}, addDisjointConcept(a_6, Person)** **β_{B_8}, addDisjointConcept(Person, a_6)**

			β_{B_6}→ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n)) and ∃i $\rho_{\exists R}(R_i, D_1)$, … , (R_i, D_n) such that ∀j $D_i \notin T$ • If ε_R(teaches,Re) and $\rho_{\exists R}(Re, De_1)$, … , (Re, De_n) such that ∀j $T \neq De_j$ if ι_R (teaches,(iD₁,iRa₁)),…,(teaches,(iDₙ,iRa₁)) and ∃i such that ∀j iRaᵢ∉T β_{B_7}→ • If $\leq_R$ (attends, R_1), … , (attends, R_n) and ∃i $\rho_{\forall R}(R_i, D_1)$, … , (R_i, D_n)) such that ∀j $D_i \notin T$ • If ε_R(attends,Re) and $\rho_{\forall R}(Re, De_1)$, … , (Re, De_n)) such that ∀j $T \neq De_j$ if ι_R(attends,(iD₁,iRa₁)),…,(attends,(iDₙ,iRa₁)) and ∃i such that ∀j iRaᵢ∉T β_{B_8}→ • If ∃i $\leq_C$ (Person) = {(Person, Di_1), … , (Person, Di_n)} and ∃j∃k $-_C$(Dij)j=(Dij,Djk) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k $-_C$(Dij,Djk) • If ∃i $\leq_C$ (Person, Di_1), … , (Person, Di_n)and ∃j∃k∃l such that δ_C(Dij, a_6) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k∃l such that δ_C(Dij, Person) • If $-_C$(Person,a_6) • If $-_C$((a_6,Person) • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_6,Ij₁),…,(a_6,Ijₙ)} and ∃k ι_C(Di-,Ijₖ) • If ∃i $-_C$(a_6,Di-) and ∃j ι_C((Person,Ij₁),…,(Person,Ijₙ)}and ∃k ι_C(Di-,Ijₖ) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ijₖ) and ∃l ι_C(Di-,I-ₗ) such that ε_I(Ijₖ, I-ₗ) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ijₖ) and ∃l ι_C(Di-,I-ₗ) such that ε_I(Ijₖ, I-ₗ) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ijₖ) and ∃l ι_C(Di-,I-ₗ) such that ε_I(I-ₗ ,Ijₖ) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ijₖ) and ∃l ι_C(Di-,I-ₗ) such that ε_I(I-ₗ) = (I-ₗ ,Ijₖ) • If ∃i $-_C$(Person)ᵢ=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i $-_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** • **If δ_C(Person, a_6)** •	
β_{B_9} =addConceptIntersection(Person, a_7)		a_9 :={⊓$_C$(Person)={Person, a_7)} ∈ **sC**	β_{B_3}→ • if $\leq_C$((Professor, $D1_1$), … , (Professor, $D1_n$)and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and ∃i such that $\leq_C$ (a_8, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(a_8, Student)** • **if ε_C(Student, Professor)** • **if ε_C(Student, a_8)** • **if ⊓$_C$(Student, Professor)** • **if ⊓$_C$(Student, a_8)** • **if ⊓$_C$(Professor, Student)** • **if ⊓$_C$(a_8, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Student, D2-) and δ_C(a_8, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if $-_C$(a_8, D1-) and δ_C(Student, D1-) • if ι_C(Student,I2₁),…,(Student, I2ₙ) and ∃i such that ι_C(Professor, I2ᵢ) • if ι_C(Student,I2₁),…,(Student, I2ₙ) and ∃i such that ι_C(a_8, I2ᵢ) • if ι_C(Professor,I1₁),…,(Professor, I1 ₙ) and ∃j such that ι_C(Student, I1 ᵢ) • if ι_C(a_8,I1₁),…,(a_8, I1 ₙ) and ∃j such that ι_C(Student, I1 ᵢ) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, I1₁),…,(Professor, I1ₙ) such that ε_I(I1ᵢ, I2ᵢ) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, I1₁),…,(a_8, I1ₙ) such that ε_I(I1ᵢ, I2ᵢ) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, I1₁),…,(Professor, I1ₙ) such that ε_I(I2 ⱼ, I1 ᵢ) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$)} and ι_C(a_8, I1₁),…,(a_8, I1ₙ) such that ε_I(I2 ⱼ, I1 ᵢ) β_{B_6}→ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n)) and ∃i $\rho_{\exists R}(R_i, D_1)$, … , (R_i, D_n) such that ∀j $D_i \notin T$ • If ε_R(teaches,Re) and $\rho_{\exists R}(Re, De_1)$, … , (Re, De_n) such that ∀j $T \neq De_j$ if ι_R (teaches,(iD₁,iRa₁)),…,(teaches,(iDₙ,iRa₁)) and ∃i such that ∀j iRaᵢ∉T β_{B_7}→	β_{B_3}, *addEquivalentConcept(Professor, Student)* β_{B_3}, *addEquivalentConcept(Student, Professor)* β_{B_3}, *addConceptIntersection(Professor, Student)* β_{B_3}, *addConceptIntersection(Student, Professor)* β_{B_8}, *addDisjointConcept(a_6, Person)* β_{B_8}, *addDisjointConcept(Person, a_6)* ***β_{B_9}, addComplementConcept(a_7, Person)*** ***β_{B_9}, addComplementConcept(Person, a_7)*** ***β_{B_9}, addDisjointConcept(a_7, Person)*** ***β_{B_9}, addDisjointConcept(Person, a_7)***

			• If $\leq_R$ (attends, R_1), … , (attends, R_n) and ∃i $\rho_{\forall R}$(R_i, D_1), … , (R_i, D_n)) such that ∀j $D_i \notin T$ • If ε_R(attends,Re) and $\rho_{\forall R}$(Re, De_1), … , (Re, De_n)) such that ∀j $T \neq De_j$ if ι_R(attends,(iD1,iRa1)),…,(attends,(iDn,iRa1)) and ∃i such that ∀j $iRa_i \notin T$ β_{B_8} → • If ∃i $\leq_C$ (Person) = {(Person, Di_1), … , (Person, Di_n)} and ∃j∃k $-_C$(Dij)j=(Dij,Djk) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k $-_C$(Dij,Djk) • If ∃i $\leq_C$ (Person, Di_1), … , (Person, Di_n)and ∃j∃k∃l such that δ_C(Dij, a_6) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k∃l such that δ_C(Dij, Person) • If $-_C$(Person,a_6) • If $-_C$(a_6,Person) • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_6,Ij_1),…,(a_6,Ij_n)} and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ι_C((Person,Ij_1),…,(Person,Ij_n)}and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l} ,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l}) = (I_{-l} ,Ij_k) • If ∃i $-_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i $-_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** • **If δ_C(Person, a_6)** β_{B_9} → • **If ∃i** $\leq_C$ (a_7, Di_1), … , (a_7, Di_n) and ∃j∃k∃l such that δ_C(Dij, Person) • **If $-_C$(Person,a_7)** • **If $-_C$(a_7,Person)** • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_7,Ij_1),…,(a_7,Ij_n) and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(a_7,Di-) and ∃j ι_C(Person,Ij_1),…,(Person,Ij_n) and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l} ,Ij_k) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l} ,Ij_k) • If ∃i $-_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_7)=(a_7, Di-) • If ∃i $-_C$(a_7,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_7, Person)** • **if δ_C(Person, a_7)** • If $-_C$((a_6,Person) • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_6,Ij_1),…,(a_6,Ij_n)} and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ι_C((Person,Ij_1),…,(Person,Ij_n)}and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(Ij_k, I_{-l}) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l} ,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ij_k) and ∃l ι_C(Di-,I_{-l}) such that ε_I(I_{-l}) = (I_{-l} ,Ij_k) • If ∃i $-_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i $-_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** **If δ_C(Person, a_6)**	
$\beta_{B_{10}}$=*addEquivalentConcept(a_8, Professor)*	addEquivalentConcept(D1 D2) • if $\leq_C$ (D2, $D2_1$), … , (D2, $D2_n$) and $-_C$(D1,D1-) and ∃j such that $\leq_C$(D1-,$D2_j$) • if$\leq_C$ (D1, $D1_1$), … , (D1, $D1_n$) and ∃i $-_C$(D2,D2-) and ∃j such that $\leq_C$(D2-,$D1_j$) • if$-_C$(D1,D1-) and ε_C(D2,D1-) • if $-_C$(D2,D2-) and ε_C(D1,D2-) • if δ_C(D1,D2)		β_{B_9} → • if $\leq_C$(Professor, $D1_1$), … , (Professor, $D1_n$)and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and ∃i such that $\leq_C$ (a_8, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)and ∃i such that $\leq_C$ (Student, $D1_i$) • **if ε_C(Professor, Student)** • **if ε_C(a_8, Student)** • **if ε_C(Student, Professor)** • **if ε_C(Student, a_8)**	β_{B_3}, *addEquivalentConcept(Professor, Student)* β_{B_3}, *addEquivalentConcept(Student, Professor)* β_{B_3}, *addConceptIntersection(Professor, Student)* β_{B_3}, *addConceptIntersection(Student, Professor)* β_{B_8}, *addDisjointConcept(a_6, Person)* β_{B_8}, *addDisjointConcept(Person, a_6)* β_{B_9}, *addComplementConcept(a_7, Person)* β_{B_9}, *addComplementConcept(Person, a_7)*

	• if δ_C(D2,D1) • if $-_C$(D1,D2) • if $-_C$(D2,D1) • if ι_C(D2,I2$_1$),…,(D2,I2$_n$) and ∃i $-_C$(D1,D1-) and ∃j such that ι_C(D1-, I2$_j$) • if ι_C(D1,I1$_1$),…,(D1,I1$_n$) and ∃i $-_C$(D2,D2-) and ∃j such that ι_C(D2-, I1$_j$) • if ∃i $-_C$(D1,D1-) and ι_C(D1-,I1-$_1$),…,(D1-,I1-$_n$) and ∃j such that ε_I(I1-$_j$, I2$_i$) • if ∃i $-_C$(D2,D2-) and ι_C(D2-,I2-$_1$),…,(D2,I2-$_n$) and ∃ j such that ε_I(I2-$_j$, I1$_i$)		• **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(Student, a_8)** • **if $\sqcap_C$(Professor, Student)** • **if $\sqcap_C$(a_8, Student)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(Student, D2-) and δ_C(a_8, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if $-_C$(a_8, D1-) and δ_C(Student, D1-) • if ι_C(Student,I2$_1$),…,(Student, I2$_n$) and ∃i such that ι_C(Professor, I2$_i$) • if ι_C(Student,I2$_1$),…,(Student, I2$_n$) and ∃i such that ι_C(a_8, I2$_i$) • if ι_C(Professor,I1$_1$),…,(Professor, I1 $_n$) and ∃j such that ι_C(Student, I1 $_i$) • if ι_C(a_8,I1$_1$),…,(a_8, I1 $_n$) and ∃j such that ι_C(Student, I1 $_i$) • if ∃i, j ι_C(Student, *I2*$_1$),…,(Student, *I2*$_n$) and ι_C(Professor, I1$_1$),…,(Professor, I1$_n$) such that ε_I(I1$_i$, I2$_j$) • if ∃i, j ι_C(Student, *I2*$_1$),…,(Student, *I2*$_n$) and ι_C(a_8, I1$_1$),…,(a_8, I1$_n$) such that ε_I(I1$_i$, I2$_j$) • if ∃i, j ι_C(Student, *I2*$_1$),…,(Student, *I2*$_n$) and ι_C(Professor, I1$_1$),…,(Professor, I1$_n$) such that ε_I(I2 $_j$, I1 $_i$) • if ∃i, j ι_C(Student, *I2*$_1$),…,(Student, *I2*$_n$)} and ι_C(a_8, I1$_1$),…,(a_8, I1$_n$) such that ε_I(I2 $_j$, I1 $_i$) β_{B_6}→ • If $\leq_R$ (teaches, R$_1$), … , (teaches, R$_n$)) and ∃i $\rho_{\exists R}$(R$_i$, D$_1$), … , (R$_i$, D$_n$) such that ∀j D$_i$ ∉ T • If ε_R(teaches,Re) and $\rho_{\exists R}$(Re, De$_1$), … , (Re, De$_n$) such that ∀j T ≠ De$_j$ if ι_R (teaches,(iD$_1$,iRa$_1$)),…,(teaches,(iD$_n$,iRa$_1$)) and ∃i such that ∀j iRa$_i$∉T β_{B_7}→ • If $\leq_R$ (attends, R$_1$), … , (attends, R$_n$) and ∃i $\rho_{\forall R}$(R$_i$, D$_1$), … , (R$_i$, D$_n$)) such that ∀j D$_i$ ∉ T • If ε_R(attends,Re) and $\rho_{\forall R}$(Re, De$_1$), … , (Re, De$_n$)) such that ∀j T ≠ De$_j$ if ι_R(attends,(iD$_1$,iRa$_1$)),…,(attends,(iD$_n$,iRa$_1$)) and ∃i such that ∀j iRa$_i$∉T β_{B_8}→ • If ∃i $\leq_C$ (Person) = {(Person, Di$_1$), … , (Person, Di$_n$)} and ∃j∃k $-_C$(Dij)j=(Dij,Djk) • If ∃i $\leq_C$ (a$_6$, Di$_1$), … , (a$_6$, Di$_n$) and ∃j∃k $-_C$(Dij,Djk) • If ∃i $\leq_C$ (Person, Di$_1$), … , (Person, Di$_n$)and ∃j∃k∃l such that δ_C(Dij, a$_6$) • If ∃i $\leq_C$ (a$_6$, Di$_1$), … , (a$_6$, Di$_n$) and ∃j∃k∃l such that δ_C(Dij, Person) • If $-_C$(Person,a_6) • If $-_C$((a_6,Person) • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_6,Ij$_1$),…,(a_6,Ij$_n$)} and ∃k ι_C(Di-,Ij$_k$) • If ∃i $-_C$(a_6,Di-) and ∃j ι_C((Person,Ij$_1$),…,(Person,Ij$_n$)}and ∃k ι_C(Di-,Ij$_k$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$) = (I-$_l$,Ij$_k$) • If ∃i $-_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i $-_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** **If δ_C(Person, a_6)** β_{B_9}→ • **If ∃i $\leq_C$** (a_7, Di$_1$), … , (a_7, Di$_n$) and ∃j∃k∃l such that δ_C(Dij, Person) • **If $-_C$(Person,a_7)** • **If $-_C$(a_7,Person)** • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_7,Ij$_1$),…,(a_7,Ij$_n$) and ∃k ι_C(Di-,Ij$_k$) • If ∃i $-_C$(a_7,Di-) and ∃j ι_C(Person,Ij$_1$),…,(Person,Ij$_n$) and ∃k ι_C(Di-,Ij$_k$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i $-_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_7)=(a_7, Di-) • If ∃i $-_C$(a_7,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_7, Person)**	*β_{B_9}, addDisjointConcept(a_7, Person)* *β_{B_9}, addDisjointConcept(Person, a_7)* ***$\beta_{B_{10}}$, addComplementConcept(Professor, a_8)*** ***$\beta_{B_{10}}$, addComplementConcept(a_8, Professor)*** ***$\beta_{B_{10}}$, addDisjointConcept(Professor, a_8)*** ***$\beta_{B_{10}}$, addDisjointConcept(a_8, Professor)*** ***β_{B_3}, $\beta_{B_{10}}$, addEquivalentConcept(a_8, Student)*** ***β_{B_3}, $\beta_{B_{10}}$, addEquivalentConcept(Student, a_8)*** ***β_{B_3}, $\beta_{B_{10}}$, addConceptIntersection(Student, a_8)*** ***β_{B_3}, $\beta_{B_{10}}$, addConceptIntersection(a_8, Student)*** ***β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addEquivalentConcept(a_6, Student)*** ***β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addEquivalentConcept(Student, a_6)*** ***β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addConceptIntersection(Student, a_6)*** ***β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addConceptIntersection(a_6, Student)***

			• **if δ_C(Person, a_7)** $\beta_{B_{10}}$→ • if $\leq_C$= (Professor, $D2_1$), … , (Professor, $D2_n$ and $-_C$(a_8,D1-) and ∃j such that $\leq_C$(D1-,$D2_i$) • if$\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and $-_C$(Professor,D2-) and ∃j such that $\leq_C$(D2-,$D1_j$) • if$-_C$(a_9,D1-) and ε_C(Professor,D1-) • if $-_C$(Professor,D2-) andj ε_C(a_8,D2-) • **if $-_C$(a_8, Professor)** • **if $-_C$(Professor, a_8)** • **if δ_C(a_8, Professor)** • **if δ_C(Professor, a_8)** • if $-_C$(a_8,D1-) and ι_C(Professor,$I2_1$),…,(Professor,$I2_n$) and ∃j ι_C(D1-, $I2_i$) • If $-_C$(a_8,D1-) and ι_C(Professor ,$I2_1$),…,(Professor ,$I2_n$) and ι_C(D1-,$I1_1$),…,(D1-,$I1_n$) and ∃j ∃k such that ε_I($I1_j$)$_i$ = ($I1_j$, $I2_k$) • if $-_C$(Professor,D2-) and ι_C(a_8,$I1_1$),…,(a_8,$I1_n$) and ∃j ι_C(D2-, $I1_j$) • if $-_C$(Professor ,D2-) and ι_C(a_8 ,$I1_1$),…,(a_8 ,$I1_n$) and ι_C(D2-,$I2_n$),…,(D2-,$I2_n$) and ∃j ∃k such that ε_I($I2_j$, $I1_k$) •	
$\beta_{B_{11}}$=*addEquivalentConcept(a_9 , Student)*			β_{B_3}→ • if $\leq_C$(Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and ∃i such that $\leq_C$ (a_8, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)and ∃i such that $\leq_C$ (Student, $D1_i$) • if $\leq_C$ (a_9, $D2_1$), … , (a_9, $D2_n$)and ∃i such that $\leq_C$ (a_9, $D1_i$) • **if ε_C (Professor, a_9)** • **if ε_CProfessor, Student)** • **if $\varepsilon_C a_8$, Student)** • **if ε_C (a_8, a_9)** • **if ε_C(Student, Professor)** • **if ε_C(a_9, Professor)** • **if ε_C((Student, a_8)** • **if ε_C(a_9, a_8)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(a_9, Professor)** • **if $\sqcap_C$((Student, a_8)** • **if $\sqcap_C$(a_9, a_8)** • **if $\sqcap_C$(Professor, Student)** • **if $\sqcap_C$(Professor, a_9)** • **if $\sqcap_C$(a_8, Student)** • **if $\sqcap_C$(a_8, a_9)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(a_9, D2-) and δ_C(Professor, D2-) • if $-_C$(Student, D2-) andj δ_C(a_8, D2-) • if $-_C$(a_9, D2-) and δ_C(a_8, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if $-_C$(Professor, D1-) and δ_C(a_9, D1-) • if $-_C$(a_8)$_i$=(a_8, D1-) and δ_C(a_9, D1-) • if $-_C$(a_8, D1-) and δ_C(a_9, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(a_9,$I2_1$),…,(a_9, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(a_8, $I2_i$) • if ι_C(a_9,$I2_1$),…,(a_9, $I2_n$) and ∃i such that $\iota_C a_8$, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(a_9, $I1_i$) • if ι_C(a_8,$I1_1$),…,(a_8, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ι_C(a_8,$I1_1$),…,(a_8, $I1_n$) and ∃j such that ι_C(a_9, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I1_i$, $I2_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$)} and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$)} such that ε_I($I1_i$, $I2_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that ε_I($I1_i$, $I2_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that ε_I($I1_i$, $I2_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_9, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$)} such that ε_I($I2_j$, $I1_i$)	β_{B_3}, *addEquivalentConcept(Professor, Student)* β_{B_3}, *addEquivalentConcept(Student, Professor)* β_{B_3}, *addConceptIntersection(Professor, Student)* β_{B_3}, *addConceptIntersection(Student, Professor)* β_{B_8}, *addDisjointConcept(a_6, Person)* β_{B_8}, *addDisjointConcept(Person, a_6)* β_{B_9}, *addDisjointConcept(a_7, Person)* β_{B_9}, *addDisjointConcept(Person, a_7)* $\beta_{B_{10}}$, *addComplementConcept(Professor, a_8)* $\beta_{B_{10}}$, *addComplementConcept(a_8, Professor)* $\beta_{B_{10}}$, *addDisjointConcept(Professor, a_8)* $\beta_{B_{10}}$, *addDisjointConcept(a_8, Professor)* β_{B_3}, $\beta_{B_{10}}$, *addEquivalentConcept(a_8, Student)* β_{B_3}, $\beta_{B_{10}}$, *addEquivalentConcept(Student, a_8)* β_{B_3}, $\beta_{B_{10}}$, *addConceptIntersection(Student, a_8)* β_{B_3}, $\beta_{B_{10}}$, *addConceptIntersection(a_8, Student)* β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, *addEquivalentConcept(a_6, Student)* β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, *addEquivalentConcept(Student, a_6)* β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, *addConceptIntersection(Student, a_6)* β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, *addConceptIntersection(a_6, Student)* ***$\beta_{B_{11}}$, addComplementConcept(a_9, Student)*** ***$\beta_{B_{11}}$, addComplementConcept(Student, a_9)*** ***$\beta_{B_{11}}$, addDisjointConcept(a_9, Student)*** ***$\beta_{B_{11}}$, addDisjointConcept(Student, a_9)*** ***β_{B_3}, $\beta_{B_{11}}$, addEquivalentConcept(a_9 Professor)*** ***β_{B_3}, $\beta_{B_{11}}$, addEquivalentConcept(Professor, a_9)*** ***β_{B_3}, $\beta_{B_{11}}$, addConceptIntersection(Professor, a_9)*** ***β_{B_3}, $\beta_{B_{11}}$, addConceptIntersection(a_9, Professor)*** ***β_{B_3},$\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_9 a_8)*** ***β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_8, a_9)*** ***β_{B_3}, $\beta_{B_{10}}$,$\beta_{B_{11}}$, addConceptIntersection(a_8, a_9)*** ***β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_9, a_8)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addEquivalentConcept(a_7, Professor)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addEquivalentConcept(Professor, a_7)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addConceptIntersection(Professor, a_7)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addConceptIntersection(a_7, Professor)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_7, a_8)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_8, a_7)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_8, a_7)*** ***β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_7, a_8)***

			$\beta_{B_6}\rightarrow$ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n)) and ∃i $\rho_{\exists R}(R_i, D_1)$, … , (R_i, D_n) such that ∀j $D_i \not\subseteq$ T • If ε_R(teaches,Re) and $\rho_{\exists R}$(Re, De_1), … , (Re, De_n) such that ∀j T ≠ De_j if ι_R (teaches,(iD_1,iRa_1)),…,(teaches,(iD_n,iRa_1)) and ∃i such that ∀j $iRa_i \notin$T $\beta_{B_7}\rightarrow$ • If $\leq_R$ (attends, R_1), … , (attends, R_n) and ∃i $\rho_{\forall R}(R_i, D_1)$, … , (R_i, D_n)) such that ∀j $D_i \not\subseteq$ T • If ε_R(attends,Re) and $\rho_{\forall R}$(Re, De_1), … , (Re, De_n)) such that ∀j T ≠ De_j if ι_R(attends,(iD_1,iRa_1)),…,(attends,(iD_n,iRa_1)) and ∃i such that ∀j $iRa_i \notin$T $\beta_{B_8}\rightarrow$ • If ∃i $\leq_C$ (Person) = {(Person, Di_1), … , (Person, Di_n)} and ∃j∃k $-_C$(Dij)j=(Dij,Djk) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k $-_C$(Dij,Djk) • If ∃i $\leq_C$ (Person, Di_1), … , (Person, Di_n)and ∃j∃k∃l such that δ_C(Dij, a_6) • If ∃i $\leq_C$ (a_6, Di_1), … , (a_6, Di_n) and ∃j∃k∃l such that δ_C(Dij, Person) • If $-_C$(Person,a_6) • If $-_C$(a_6,Person) • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_6,Ij_1),…,(a_6,Ij_n)} and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ι_C((Person,Ij_1),…,(Person,Ij_n)}and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I(Ij_k, $I_{\neg l}$) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I(Ij_k, $I_{\neg l}$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I($I_{\neg l}$,Ij_k) • If ∃i $-_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I($I_{\neg l}$) = ($I_{\neg l}$,Ij_k) • If ∃i $-_C$(Person)$_l$=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i $-_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** • **If δ_C(Person, a_6)** $\beta_{B_9}\rightarrow$ • **If** ∃i $\leq_C$ (a_7, Di_1), … , (a_7, Di_n) and ∃j∃k∃l such that δ_C(Dij, Person) • **If $-_C$(Person,a_7)** • **If $-_C$(a_7,Person)** • If ∃i $-_C$(Person,Di-) and ∃j ι_C(a_7,Ij_1),…,(a_7,Ij_n) and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(a_7,Di-) and ∃j ι_C(Person,Ij_1),…,(Person,Ij_n) and ∃k ι_C(Di-,Ij_k) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I(Ij_k, $I_{\neg l}$) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I(Ij_k, $I_{\neg l}$) • If ∃i $-_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I($I_{\neg l}$,Ij_k) • If ∃i $-_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij_k) and ∃l ι_C(Di-,$I_{\neg l}$) such that ε_I($I_{\neg l}$,Ij_k) • If ∃i $-_C$(Person)$_l$=(Person,Di-) and ∃j ε_C(a_7)=(a_7, Di-) • If ∃i $-_C$(a_7,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_7, Person)** • **if δ_C(Person, a_7)** $\beta_{B_{10}}\rightarrow$ • if $\leq_C$= (Professor, $D2_1$), … , (Professor, $D2_n$ and $-_C$(a_8,D1-) and ∃j such that $\leq_C$(D1-,$D2_j$) • if$\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and $-_C$(Professor,D2-) and ∃j such that $\leq_C$(D2-,$D1_j$) • if$-_C$(a_8,D1-) and ε_C(Professor,D1-) • if $-_C$(Professor,D2-) andj ε_C(a_8,D2-) • **if $-_C$(a_8, Professor)** • **if $-_C$(Professor, a_8)** • **if δ_C(a_8, Professor)** • **if δ_C(Professor, a_8)** • if $-_C$(a_8,D1-) and ι_C(Professor,$I2_1$),…,(Professor,$I2_n$) and ∃j ι_C(D1-, $I2_j$) • If $-_C$(a_8,D1-) and ι_C(Professor ,$I2_1$),…,(Professor ,$I2_n$) and ι_C(D1-,$I1_{\neg 1}$),…,(D1-,$I1_{\neg n}$) and ∃j ∃k such that ε_I($I1_{\neg j}$)$_j$ = ($I1_{\neg j}$, $I2_k$) • if $-_C$(Professor,D2-) and ι_C(a_8,$I1_1$),…,(a_8,$I1_n$) and ∃j ι_C(D2-, $I1_j$) • if $-_C$(Professor ,D2-) and ι_C(a_8 ,$I1_1$),…,(a_8 ,$I1_n$) and ι_C(D2-,$I2_{\neg 1}$),…,(D2-,$I2_{\neg n}$) and ∃j ∃k such that ε_I($I2_{\neg j}$, $I1_k$) $\beta_{B_{11}}\rightarrow$	

			• if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$ and $-_C$(a_9,D1-) and ∃j such that $\leq_C$(D1-,$D2_i$) • if $\leq_C$ (a_9, $D1_1$), … , (a_9, $D1_n$) and $-_C$(Student,D2-) and ∃j such that $\leq_C$(D2-,$D1_i$) • if $-_C$(a_9,D1-) andε_C(Student,D1-) • if ∃i $-_C$(Student,D2-) andε_C(a_9,D2-) • **if $-_C$(a_9, Student)** • **if$-_C$ (Student, a_9)** • **if δ_C(a_9, Student)** • **if δ_C(Student, a_9)** • If $-_C$(a_9,D1-) and ι_C(Student,$I2_1$),…,(Student,$I2_n$) and ∃j ι_C(D1-, $I2_i$) • If $-_C$(a_9,D1-) and ι_C(Student ,$I2_1$),…,(Student ,$I2_n$) and ι_C(D1-,$I1_1$-),…,(D1-,$I1_n$-) and ∃j ∃k such that ε_I($I1_j$-, $I2_k$) • If $-_C$(Student,D2-) and ι_C(a_9,$I1_1$),…,(a_9,$I1_n$) and ∃j ι_C(D2-, $I1_j$) • If $-_C$(Student ,D2-) and ι_C(a_9 ,$I1_1$),…,(a_9 ,$I1_n$) and ι_C(D2-,$I2_1$-),…,(D2-,$I2_n$-) and ∃j ∃k such that ε_I($I2_j$-, $I1_k$)	
$\beta_{B_{12}}$=*addConcept(PhDStudent)*	addConcept(D)	*PhDStudent* ∈ *sC*	β_{B_3}→ • if $\leq_C$(Professor, $D1_1$), … , (Professor, $D1_n$) and ∃i such that $\leq_C$ (Professor, $D2_i$) • if $\leq_C$ (a_8, $D1_1$), … , (a_8, $D1_n$) and ∃i such that $\leq_C$ (a_8, $D2_i$) • if $\leq_C$ (Student, $D2_1$), … , (Student, $D2_n$)and ∃i such that $\leq_C$ (Student, $D1_i$) • if $\leq_C$ (a_9, $D2_1$), … , (a_9, $D2_n$)and ∃i such that $\leq_C$ (a_9, $D1_i$) • **if ε_C (Professor, a_9)** • **if ε_CProfessor, Student)** • **if $\varepsilon_C a_8$, Student)** • **if ε_C (a_8, a_9)** • **if ε_C(Student, Professor)** • **if ε_C(a_9, Professor)** • **if ε_C((Student, a_8)** • **if ε_C(a_9, a_8)** • **if $\sqcap_C$(Student, Professor)** • **if $\sqcap_C$(a_9, Professor)** • **if $\sqcap_C$((Student, a_8)** • **if $\sqcap_C$(a_9, a_8)** • **if $\sqcap_C$(Professor, Student)** • **if $\sqcap_C$(Professor, a_9)** • **if $\sqcap_C$(a_8, Student)** • **if $\sqcap_C$(a_8, a_9)** • if $-_C$(Student, D2-) and δ_C(Professor, D2-) • if $-_C$(a_9, D2-) and δ_C(Professor, D2-) • if $-_C$(Student, D2-) and δ_C(a_8, D2-) • if $-_C$(a_9, D2-) and ∃j δ_C(a_8, D2-) • if $-_C$(Professor, D1-) and δ_C(Student, D1-) • if $-_C$(Professor, D1-) and δ_C(a_9, D1-) • if $-_C$(a_8)$_i$=(a_8, D1-) and δ_C(a_9, D1-) • if $-_C$(a_8, D1-) and δ_C(a_9, D1-) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(a_9,$I2_1$),…,(a_9, $I2_n$) and ∃i such that ι_C(Professor, $I2_i$) • if ι_C(Student,$I2_1$),…,(Student, $I2_n$) and ∃i such that ι_C(a_8, $I2_i$) • if ι_C(a_9,$I2_1$),…,(a_9, $I2_n$) and ∃i such that $\iota_C a_8$, $I2_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ι_C(Professor,$I1_1$),…,(Professor, $I1_n$) and ∃j such that ι_C(a_9, $I1_i$) • if ι_C(a_8,$I1_1$),…,(a_8, $I1_n$) and ∃j such that ι_C(Student, $I1_i$) • if ι_C(a_8,$I1_1$),…,(a_8, $I1_n$) and ∃j such that ι_C(a_9, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$)} and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$)} such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that ε_I($I1_i$, $I2_j$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(Professor, $I1_1$),…,(Professor, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(Student, $I2_1$),…,(Student, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$) such that ε_I($I2_j$, $I1_i$) • if ∃i, j ι_C(a_9, $I2_1$),…,(a_9, $I2_n$) and ι_C(a_8, $I1_1$),…,(a_8, $I1_n$)} such that ε_I($I2_j$, $I1_i$) β_{B_6}→ • If $\leq_R$ (teaches, R_1), … , (teaches, R_n)) and ∃i $\rho_{\exists R}$(R_i, D_1), … , (R_i, D_n) such that ∀j $D_i \not\subseteq T$ • If ε_R(teaches,Re) and $\rho_{\exists R}$(Re, De_1), … , (Re, De_n) such that ∀j $T \neq De_i$	*β_{B_3}, addEquivalentConcept(Professor, Student)* *β_{B_3}, addEquivalentConcept(Student, Professor)* *β_{B_3}, addConceptIntersection(Professor, Student)* *β_{B_3}, addConceptIntersection(Student, Professor)* *β_{B_8}, addDisjointConcept(a_6, Person)* *β_{B_8}, addDisjointConcept(Person, a_6)* *β_{B_9}, addDisjointConcept(a_7, Person)* *β_{B_9}, addDisjointConcept(Person, a_7)* *$\beta_{B_{10}}$, addComplementConcept(Professor, a_8)* *$\beta_{B_{10}}$, addComplementConcept(a_8, Professor)* *$\beta_{B_{10}}$, addDisjointConcept(Professor, a_8)* *$\beta_{B_{10}}$, addDisjointConcept(a_8, Professor)* *β_{B_3}, $\beta_{B_{10}}$, addEquivalentConcept(a_8, Student)* *β_{B_3}, $\beta_{B_{10}}$, addEquivalentConcept(Student, a_8)* *β_{B_3}, $\beta_{B_{10}}$, addConceptIntersection(Student, a_8)* *β_{B_3}, $\beta_{B_{10}}$, addConceptIntersection(a_8, Student)* *β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addEquivalentConcept(a_6, Student)* *β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addEquivalentConcept(Student, a_6)* *β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addConceptIntersection(Student, a_6)* *β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, addConceptIntersection(a_6, Student)* *$\beta_{B_{11}}$, addComplementConcept(a_9, Student)* *$\beta_{B_{11}}$, addComplementConcept(Student, a_9)* *$\beta_{B_{11}}$, addDisjointConcept(a_9, Student)* *$\beta_{B_{11}}$, addDisjointConcept(Student, a_9)* *β_{B_3}, $\beta_{B_{11}}$, addEquivalentConcept(a_9 Professor)* *β_{B_3}, $\beta_{B_{11}}$, addEquivalentConcept(Professor, a_9)* *β_{B_3}, $\beta_{B_{11}}$, addConceptIntersection(Professor, a_9)* *β_{B_3}, $\beta_{B_{11}}$, addConceptIntersection(a_9, Professor)* *β_{B_3},$\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_9 a_8)* *β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_8, a_9)* *β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_8, a_9)* *β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_9, a_8)* *β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addEquivalentConcept(a_7, Professor)* *β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addEquivalentConcept(Professor, a_7)* *β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addConceptIntersection(Professor, a_7)* *β_{B_3}, β_{B_9}, $\beta_{B_{11}}$, addConceptIntersection(a_7, Professor)* *β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_7, a_8)* *β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addEquivalentConcept(a_8, a_7)* *β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_8, a_7)* *β_{B_3}, β_{B_9}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, addConceptIntersection(a_7, a_8)*

			if ι_R (teaches,(iD$_1$,iRa$_1$)),...,(teaches,(iD$_n$,iRa$_1$)) and ∃i such that ∀j iRa$_i$∉T β_{B_7}→ • If $\leq_R$ (attends, R$_1$), … , (attends, R$_n$) and ∃i $\rho_{\forall R}$(R$_i$, D$_1$), … , (R$_i$, D$_n$)) such that ∀j D$_i$ ⋤ T • If ε_R(attends,Re) and $\rho_{\forall R}$(Re, De$_1$), … , (Re, De$_n$)) such that ∀j T ≠ De$_j$ if ι_R(attends,(iD$_1$,iRa$_1$)),...,(attends,(iD$_n$,iRa$_1$)) and ∃i such that ∀j iRa$_i$∉T β_{B_8}→ • If ∃i $\leq_C$ (Person) = {(Person, Di$_1$), … , (Person, Di$_n$)} and ∃j∃k −$_C$(Dij)j=(Dij,Djk) • If ∃i $\leq_C$ (a$_6$, Di$_1$), … , (a$_6$, Di$_n$) and ∃j∃k −$_C$(Dij,Djk) • If ∃i $\leq_C$ (Person, Di$_1$), … , (Person, Di$_n$)and ∃j∃k∃l such that δ_C(Dij, a$_6$) • If ∃i $\leq_C$ (a$_6$, Di$_1$), … , (a$_6$, Di$_n$) and ∃j∃k∃l such that δ_C(Dij, Person) • If −$_C$(Person,a_6) • If −$_C$(a_6,Person) • If ∃i −$_C$(Person,Di-) and ∃j ι_C(a_6,Ij$_1$),...,(a_6,Ij$_n$)} and ∃k ι_C(Di-,Ij$_k$) • If ∃i −$_C$(a_6,Di-) and ∃j ι_C((Person,Ij$_1$),...,(Person,Ij$_n$)}and ∃k ι_C(Di-,Ij$_k$) • If ∃i −$_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i −$_C$(a_6,Di-) and ∃j ∃k ι_C(a_6,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i −$_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i −$_C$(a_6,Di-) and ∃j ∃k $\iota_C a_6$,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$) = (I-$_l$,Ij$_k$) • If ∃i −$_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_6)=(a_6, Di-) • If ∃i −$_C$(a_6,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_6, Person)** **If δ_C(Person, a_6)** β_{B_9}→ • **If** ∃i $\leq_C$ (a_7, Di$_1$), … , (a_7, Di$_n$) and ∃j∃k∃l such that δ_C(Dij, Person) • **If −$_C$(Person,a_7)** • **If −$_C$(a_7,Person)** • If ∃i −$_C$(Person,Di-) and ∃j ι_C(a_7,Ij$_1$),...,(a_7,Ij$_n$) and ∃k ι_C(Di-,Ij$_k$) • If ∃i −$_C$(a_7,Di-) and ∃j ι_C(Person,Ij$_1$),...,(Person,Ij$_n$) and ∃k ι_C(Di-,Ij$_k$) • If ∃i −$_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i −$_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(Ij$_k$, I-$_l$) • If ∃i −$_C$(Person,Di-) and ∃j ∃k ι_C(Person,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i −$_C$(a_7,Di-) and ∃j ∃k ι_C(a_7,Ij$_k$) and ∃l ι_C(Di-,I-$_l$) such that ε_I(I-$_l$,Ij$_k$) • If ∃i −$_C$(Person)$_i$=(Person,Di-) and ∃j ε_C(a_7)=(a_7, Di-) • If ∃i −$_C$(a_7,D-) and ∃j ε_C(Person, Di-) • **If δ_C(a_7, Person)** • **if δ_C(Person, a_7)** $\beta_{B_{10}}$→ • if $\leq_C$= (Professor, D2$_1$), … , (Professor, D2$_n$ and −$_C$(a_8,D1-) and ∃j such that $\leq_C$(D1-,D2$_i$) • if $\leq_C$ (a_8, D1$_1$), … , (a_8, D1$_n$) and −$_C$(Professor,D2-) and ∃j such that $\leq_C$(D2-,D1$_j$) • if−$_C$(a_8,D1-) and ∃j ε_C(Professor,D1-) • if −$_C$(Professor,D2-) and ∃j ε_C(a_8,D2-) • **if −$_C$(a_8, Professor)** • **if −$_C$(Professor, a_8)** • **if δ_C(a_8, Professor)** • **if δ_C(Professor, a_8)** • if −$_C$(a_8,D1-) and ι_C(Professor,I2$_1$),...,(Professor,I2$_n$) and ∃j ι_C(D1-, I2$_i$) • If −$_C$(a_8,D1-) and ι_C(Professor ,I2$_1$),...,(Professor ,I2$_n$) and ι_C(D1-,I1-$_1$),...,(D1-,I1-$_n$) and ∃j ∃k such that ε_I(I1-$_j$)$_i$ = (I1-$_j$, I2$_k$) • if −$_C$(Professor,D2-) and ι_C(a_8,I1$_1$),...,(a_8,I1$_n$) and ∃j ι_C(D2-, I1$_i$) • if −$_C$(Professor ,D2-) and ι_C(a_8 ,I1$_1$),...,(a_8 ,I1$_n$) and ι_C(D2-,I2-$_1$),...,(D2-,I2-$_n$) and ∃j ∃k such that ε_I(I2-$_j$, I1$_k$) $\beta_{B_{11}}$→ • if $\leq_C$ (Student, D2$_1$), … , (Student, D2$_n$and −$_C$(a_9,D1-) and ∃j such that $\leq_C$(D1-,D2$_i$) • if $\leq_C$ (a_9, D1$_1$), … , (a_9, D1$_n$) and −$_C$(Student,D2-) and ∃j such that $\leq_C$(D2-,D1$_i$) • if −$_C$(a_9,D1-) andε_C(Student,D1-)	

			• if $\exists i -_C$(Student,D2-) and$\varepsilon_C(a_9$,D2-) • **if $-_C(a_9$, Student)** • **if$-_C$ (Student, a_9)** • **if $\delta_C(a_9$, Student)** • **if δ_C(Student, a_9)** • If $-_C(a_9$,D1-) and ι_C(Student,I2$_1$),…,(Student,I2$_n$) and $\exists$j ι_C(D1-, I2$_i$) • If $-_C(a_9$,D1-) and ι_C(Student ,I2$_1$),…,(Student ,I2$_n$) and ι_C(D1-,I1-$_1$),…,(D1-,I1-$_n$) and $\exists$j $\exists$k such that ε_I(I1-$_j$, I2$_k$) • If $-_C$(Student,D2-) and $\iota_C(a_9$,I1$_1$),…,(a_9,I1$_n$) and $\exists$j ι_C(D2-, I1$_j$) • If $-_C$(Student ,D2-) and $\iota_C(a_9$,I1$_1$),…,(a_9 ,I1$_n$) and ι_C(D2-,I2-$_1$),…,(D2-,I2-$_n$) and $\exists$j $\exists$k such that ε_I(I2-$_j$, I1$_k$)	
$\beta_{B_{13}}$ =*addConceptIntersection(Student, a6)*	addConceptIntersection(D1, D2) • If $\exists i \leq_C$ (Di, Di$_1$), … , (Di, Di$_n$) and If $\exists j -_C$(Dij,Djj) • If $\exists i \leq_C$ (Di, Di$_1$), … , (Di, Di$_n$) and $\exists j \exists k \exists l$ such that δ_C(Dij,Dkl) • If $\exists i \exists j -_C$(Di,Dj) • If $\exists i -_C$(Di,Di-) and $\exists j \iota_C$(Dj,Ij1),…,(Dj,Ijn) and $\exists k \iota_C$(Di-,Ijk) • If $\exists i -_C$(Di,Di-) and $\exists j \exists k \iota_C$(Dj,Ijk) and $\exists l \iota_C$(Di-,I-l) such that ε_I(Ijk, I-l) • If $\exists i -_C$(Di,Di-) and $\exists j \exists k \iota_C$(Dj,Ijk) and $\exists l \iota_C$(Di-,I-l) such that ε_I(I-l ,Ijk) • If $\exists i -_C$(Di,Di-) and $\exists j \varepsilon_C$(Dj, Di-) • If $\exists i \exists j \delta_C$(Di, Dj)	*a_{13} :={$\sqcap_C$(Student)={Student, a_6)} $\in$ sC*	**Inconsistency Identified** **β_{B_3}, β_{B_8}, $\beta_{B_{10}}$, *addConceptIntersection(Student, a_6)***	
$\beta_{B_{14}}$ =*addSubConcept(a_{13}, PhDStudent)*	addSubConcept(D1, D2) • If $-_C$(D1,D1-) and $\leq_C$(D1-,D1-$_1$),…,(D1-,D1-$_n$) and $\exists j \exists k$ such that ε_C(D2, D1-$_j$) • if $-_C$(D1,D1-) and $\exists j \varepsilon_C$(D2, D1-) • if δ_C(D1,D2) • if δ_C(D2, D1) • if $-_C$(D1,D1-) and $\exists j \leq_C$(D1-, D2) • If $-_C$(D1)$_i$=(D1,D1-) and ι_C(D2,I2$_1$),…,(D2,I2$_n$)} and $\exists$j ι_C(D1-, I2$_i$) • If $-_C$(D1,D1-) and ι_C(D2,I2$_1$),…,(D2,I2$_n$) and ι_C(D1-,I1-$_1$),…,(D1-,I1-$_n$) and $\exists j \exists k$ such that ε_I(I1-$_j$, I2$_k$)			

Table 9. Logical Consistency Preventive Analysis Process Results Example

3.3.2 Logical Consistency Resolution Example

The minimal inconsistent set of changes detected in *log1* and returned to the domain experts is the set *{* β_{B_3} , β_{B_8} , $\beta_{B_{10}}$, $\beta_{B_{11}}$ *}={addDisjointConcept(Professor, Student), addConceptIntersection(Person,* a_6*), addEquivalentConcept(* a_8 *, Professor), addConceptIntersection(Student,* a_6*)}.* The change which has provoked the logical inconsistency is the last $\beta_{B_{11}}$= *addConceptIntersection(Student,* a_6*).* A new change modelling phase is launched to update the list of changes to apply of *log1*. If they envision a belief contraction resolution type, domain experts can then choose between deleting this change or any of the other figuring in the set. Deleting β_{B_3}or deleting $\beta_{B_{10}}$or deleting $\beta_{B_{11}}$ from *log1* are sufficient solutions to ensure a logical consistent application of the changes. However deleting only β_{B_8}, which adds the anonymous concept defined by the axiom a_8 appearing in parameter of $\beta_{B_{10}}$, violates the structural consistency of the changes. Also, if β_{B_8} is not kept by domain users during the new change modelling phase, they wil not be able to model again the $\beta_{B_{10}}$change as such. Actually, the anonymous concept defined by a_8 will not be available anymore. β_{B_8} must be removed with $\beta_{B_{10}}$ to preserve both structural and logical consistency. If a belief revision resolution type is preferred by domain experts, a sufficient solution is to add to *log1* the basic deletion change corresponding to one of the changes among β_{B_3}, $\beta_{B_{10}}$, $\beta_{B_{11}}$, i.e. respectively *deleteDisjointConcept(Professor, Student)* or *addEquivalentConcept(* a_8 *, Professor)* or *deleteConceptIntersection(Student,* a_6*)}.* For the same reason as belief contraction, the basic deletion change corresponding to β_{B_8}, i.e. *deleteConceptIntersection(Person,* a_6*),* has to be jointly added with the one corresponding to $\beta_{B_{10}}$, i.e. *deleteEquivalentConcept(*a_8*, Professor*). For both resolution approaches, supplementary changes can be added by domain experts to satisfy the ontology needs of change.

3.4 Discussion

This third section has illustrated the **logical consistency preventive resolution approach** used to help the user identifying and debugging logically inconsistent sets of changes modelled before applying them on the ontology. The approach is based on axiom unsatisfiability constraints derived from $\mathcal{SHOIN}(\mathcal{D})$ basic change operator types **logical consequences**. For each basic change operator type, the logical consequences are studied and translated into a list of **minimal inconsistent sets of generic axioms**, which could potentially be part of an ontology and cause logical inconsistencies if a change of this type is applied on the ontology. They are used to generate the **minimal inconsistent sets of changes** that need to be checked in the log of changes during the change semantics phase of an ontology evolution process. Despite the care taken in the manufacture of these inconsistent sets of axioms derived from each change operator type logical consequences, the list may not be exhaustive to be 100% certain no logical consistency is missed. The list of unsatisfiability constraints of the whole basic change operator types, presented in Annex 1, is in a work-in-progress state and requires further improvement. The logical consistency preventive approach has been illustrated through a simple example of inconsistency detection and resolution. It is important to notice that the list of minimal inconsistent sets of changes generated by the detection process is recursively grown according to the addition of new unsatisfiability constraints and of new ontological elements, brought by the successive changes. The $\mathcal{SHOIN}(\mathcal{D})$ DL having a *ExpTime* complexity for consistency checking, the number of minimal inconsistent sets of changes to check indeed grows exponentially. Also as stated in Chapter 4, the logical consistency maintenance is reinforced by a second check during the **change implementation phase** by a corrective approach using the Pellet reasoner to check the evolved copy of the ontology.

4 Conclusion

This chapter has presented the **structural** and **logical consistency preventive approaches** proposed in the OntoVersionGraph methodology. It completes the first block of the approach presented in my thesis. The structural consistency preventive approach assesses the structural compliance of the changes modelled by the user during the **change modelling phase** of the ontology evolution process. If a structural inconsistency is found, the list of changes is automatically completed with missing changes to recover structural consistency. The logical preventive approach evaluates the logical compliance of the changes structurally assessed with logical constraints issued from the ontology whole sets of axioms. It intervenes at the **change semantic phase** of the ontology evolution process. If a logical inconsistency is found, the corresponding inconsistent sets of changes are identified so the user can easily modify the list of changes to apply on the ontology. For each approach, an example is given to illustrate their application. The second block of my thesis approach, consisting in the **logical specification and extraction of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**, is presented in the next chapter.

References

Baader, F., Horrocks, I., & Sattler, U. (2004). Description logics. In Handbook on ontologies (pp. 3-28). Springer Berlin Heidelberg.

Flouris, G. (2006). On belief change in ontology evolution: Thesis. AI Communications, 19(4), 395-397.

Flouris, G., Plexousakis, D., & Antoniou, G. (2005). On applying the AGM theory to DLs and OWL. In The Semantic Web–ISWC 2005 (pp. 216-231). Springer Berlin Heidelberg.

Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In The Semantic Web-ISWC 2003 (pp. 17-29). Springer Berlin Heidelberg.

Pan, Z. (2005, November). Benchmarking DL Reasoners Using Realistic Ontologies. In OWLED (Vol. 188).

Parsia, B., & Sirin, E. (2004, November). Pellet: An owl dl reasoner. In Third International Semantic Web Conference-Poster (Vol. 18).

Ribeiro, M. M., & Wassermann, R. (2009b). Base revision for ontology debugging. Journal of Logic and Computation, 19(5), 721-743.

Ribeiro, M., Wassermann, R., Antoniou, G., Flouris, G., & Pan, J. (2009, September). Belief contraction in web-ontology languages. In Proceedings of the 3rd International Workshop on Ontology Dynamics, IWOD.

Rogozan, C. D. (2009). Gestion de l'évolution des ontologies: méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions d'ontologie (Doctoral dissertation, Université du Québec à Montréal).

Stojanovic, L. (2004). Methods and tools for ontology evolution.

Chapter 6

$\mathcal{SHOIN}(\mathcal{D})$ Ontology Views Extraction & Management

Summary

This chapter aims formalizes the block (2) of my thesis proposal, i.e. modelling the **logical extraction and change management of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. A first section presents and explains the approach for specifying views on an ontology, according to the following criteria: a view must produce a sub-ontology, so it must describe a smaller domain than the ontology one, while remaining evolvable, versionable and consistent. A second section formalizes the process of **sub-ontology extraction** from a $\mathcal{SHOIN}(\mathcal{D})$ ontology. It also shows how the global log of change of an extracted sub-ontology is built from the ontology global log of changes to support the OCM features.

Plan

1 SHOIND View Extraction Approach 148
1.1 Preliminary Definitions 148
1.2 Extraction Structural Constraints 149
1.3 Extraction Levels 151
1.4 Discussion 152
2 View Extraction Formalization 153
2.1 SHOIN(D) View Extraction 153
2.2 Starter Extraction rules 154
2.3 Complete Extraction rules 154
2.4 Partial extraction rules 159
2.5 Sub-ontology Global Log of Changes Generation Rule 160
2.6 View Extraction Example 161
2.7 View Extraction Result 162
2.8 View and Ontology Global Logs of Changes 167
2.9 Discussion 168
3 Conclusion 170

Previous chapter has completed the first block of my thesis proposal. It has presented the structural and logical consistency preventive approaches respectively used to maintain the structural and logical consistency of a $\mathcal{SHOIN}(\mathcal{D})$ ontology before changes are applied on it. They are respectively handled to validate the change modelling phase and the change semantics phase of an ontology evolution process. They allow guiding the domain experts applying consistent changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology. The change implementation phase can therefore be realized and validated by a second logical consistent check with the Pellet reasoner (Parsia & Sirin, 2004*)*. The change propagation phase is the last phase requiring a specific approach in the OntoVersionGraph methodology, as it has to assess the impact and spread the change implemented to the ontology views and/or dependent ontologies. The change impact management approach, completing the third block of the my thesis proposal, is formalized in Chapter 7. Before, it requires the formalization of the **second block**, consisting in modeling the **logical extraction and change management of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. In OntoVersionGraph, a **view on an ontology defines a sub-ontology** extracted from a $\mathcal{SHOIN}(\mathcal{D})$ ontology. A view sub-ontology is then itself defined as a $\mathcal{SHOIN}(\mathcal{D})$ ontology so it has to satisfy several criteria. It should describe a domain, be a consistent ontology, and be able to evolve and be versioned.

First, the **domain has to be defined** and **should be more specific** than the domain of the source ontology describes. In the literature, "**mappings**" are used to specify what to keep from the source ontology (c.f. Orbst & al, 2003, Noy & Musen, 2004, Seidenberg & Rector, 2006). They delimit the domain the sub-ontology will describe when extracted. In this approach, we use the term "**view definition**" to refer to a "mapping" that specify the ontological elements to keep in the view. Even if it is not a change as such, i.e. modifying the source ontology, the view definition has to be traced like a change in the source ontology global log of changes in order to display versioning features and change propagation from the ontology the view.

Second, the extraction of the view should produce a **consistent $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology** from the source ontology. Structural consistency can be ensured by an **extraction process preserving the dependencies of the extracted axioms** (c.f. Structural Consistency in Chapter 5). Concerning logical consistency, because of the monotonicity property of the $\mathcal{SHOIN}(\mathcal{D})$ DL, an ontology can only become logically inconsistent by adding axioms (c.f. Logical Consistency in Chapter 5). If a set of axioms is satisfiable, it will still be satisfiable when any axiom is deleted. Consequently the **source ontology set of axioms has to be checked as logically consistent** and a view on this ontology has to be extracted as a subset from this set, in order to produce a logically consistent sub-ontology.

Third, the sub-ontology produced should be able to **evolve and be versioned like any ontology**. Indeed, as the sub-ontology is defined as a $\mathcal{SHOIN}(\mathcal{D})$ consistent sub-portion of an ontology, describing a more specific domain than the source ontology, it may also need to evolve according to this domain or according to its users assumptions. As this sub-portion corresponds to a subset of axioms extracted from the source ontology set of axioms, its **global log of changes can be defined as a subset of the global log of changes of the source ontology**. Each view can therefore take advantage of change management features.

This chapter is devoted to provide a formal definition of this idea. The first section defines a view extraction process dedicated to $\mathcal{SHOIN}(\mathcal{D})$ formal ontologies. The second section presents a formalization of the whole view extraction process of our proposal.

1 $\mathcal{SHOIN}(\mathcal{D})$ View Extraction Approach

This section presents the approach used to define a **view extraction process** according to the criteria presented above. A first part introduces the **preliminary definitions** that will be used to define the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process. A second part focuses on the **structural constraints** ensuring the structural consistency of each subset extracted to build the view sub-ontology. A third part presents the **different extraction levels** used to optimize each subset extraction. A fourth part deals with the OntoVersionGraph methodology **evolution and versioning features** use to manage such views.

1.1 Preliminary Definitions

This part gives the preliminary definitions that will be used to define the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process, such as: the view definition on an ontology, a sub-ontology and its extraction according to a view definition.

Delimiting a sub-domain on an ontology domain can be done by manually selecting the ontological elements required in the sub-domain. However the number of ontological elements can be huge if the domain is wide and the ontology complex. Also, domain experts may know the entire terminology to be able to select the right elements to pick. According to (Noy & Musen, 2004, Seidenberg & Rector, 2006), a practical approach is to focus on **main elements** required in the sub-domain. A view definition should then only contain the main elements of the subdomain. The rest of the axioms should be automatically added to grow the sub-ontology until it meets the domain experts requirements. In the view definition approach proposed in OntoVersionGraph, main ontological elements can be: **concepts, datatypes, roles, attributes, instance** or **datavalues** according to the $\mathcal{SHOIN}(\mathcal{D})$ ontology model. A view definition is defined as below:

Definition 13: A **view definition** $V(\mathcal{S}_0)$ is a mapping between an ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$ and its sub-ontology $\mathcal{S}_0'=(\Omega_0', \Sigma_0', \Phi_0', E_0)$, which refers axioms of $\mathcal{S}_0$ to axioms of $\mathcal{S}_0'$, by an **extraction relation** specifying the main elements to keep in the corresponding **extractions**.

According to the monotonicity property of the $\mathcal{SHOIN}(\mathcal{D})$ DL, the sub-ontology is consistent, only if it corresponds to a subset of the ontology set of axioms, and if this ontology is consistent. So to ensure the sub-ontology consistency, the extraction process of a view must be equivalent to a deletion of **axioms** and not just a deletion of ontological elements from a consistent ontology. Therefore, the view extraction process can take a set of **ontological elements as input** but must produce a **set of axioms**. However it is difficult to represent and trace a view as an ontology in our approach if only defined as the result of a set of axiom deletions applied on an ontology. A view has to be defined like an ontology respecting the $\mathcal{SHOIN}(\mathcal{D})$ ontology model to take advantage of the change management features of OntoVersionGraph, i.e. a **set of** $\mathcal{SHOIN}(\mathcal{D})$ **axioms** (c.f. Definition 14).

Definition 14: A **view** V defines a sub-ontology $\mathcal{S}_0'$ extracted from an ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$ as a subset of $\mathcal{S}_0$ set of axioms, such that $\mathcal{S}_0' \subseteq \mathcal{S}_0$.

It means that a view sub-ontology can be defined as a set of axioms corresponding to a subset of axioms of the ontology, which are not deleted from the ontology. Therefore, extracting such a view comes down to extract axioms depending on ontological elements that are specified in the view definition (c.f. Definition 15).

Definition 15: The **view definition** defining the extraction **of a view** V from an ontology $\mathcal{S}_0=(\Omega_0, \Sigma_0, \Phi_0, E_0)$, noted $V(\mathcal{S}_0)$, is a set of extractions $E(\varepsilon_i)$ of axioms $a_i \in (\Omega_0, \Sigma_0, \Phi_0, E_0)$ dependant to ontological

elements $\varepsilon_i \in \{sC, sT, sR, sA, sI\ sV\} \subseteq \Omega_O$, w.r.t the *Axiom Dependency Property*, such that $V(\mathcal{S}_O)=\bigcup_{i=1}^{n} E(\varepsilon_i)$ and $E(\varepsilon_i)=\{a_1:=\{\pi_1(\varepsilon_i)=(\varepsilon_i,\ldots^*)\}, \ldots, \{a_m:=\{\pi_m(\varepsilon_i)=(\varepsilon_i,\ldots^*)\}\}$, with $^* \in \{sC, sT, sR, sA, sI, sV\} \subseteq \Omega_O$ and $i, n, m \in \mathbb{N}$.

Ontological elements ε_i requested by extractions $E(\varepsilon_i)$ are called **starter elements** in the rest of the document like in (Noy & Musen, 2004) proposal (c.f. Definition 17). "Starter" stands for the fact that they are the inputs of an extraction of axioms. It is the departure point of the extraction traversal within the ontology.

Definition 16: A **starter element** $\varepsilon_i \in \{sC, sR, sA, sI, sT, sV\} \subseteq \Omega_O$ is the main element of an extraction result.

The following part presents the structural constraints that the extraction of a view has to respect in order to ensure the consistency of the resulting view.

1.2 Extraction Structural Constraints

This part defines the structural constraints used to ensure the structural consistency of each extraction. Being designed to extract $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies, the specification of views is based on $\mathcal{SHOIN}(\mathcal{D})$ structural constraints. Like for the structural consistency preventive resolution approach, these structural constraints are based on the *Axiom Dependency* (c.f. Property 1 in Chapter 5). Definition 17 below describes their use according to starter element extractions.

Definition 17: $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraint for Starter Element Extraction. A structural constraint applying to the extraction $E(\varepsilon_i)_i$ of a starter element $\varepsilon_i \in \{sC, sT, sR, sA, sI, sV\} \subseteq \Omega_O$ from an ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$ defines a dependency between ε_i and the axioms of $\mathcal{S}_O$ for which ε_i exists as a parameter.

In other words, the use of structural constraints allows **preserving the structural context of the starter element** extracted in the subset. Therefore real sub-portions of the ontology can be extracted, as it is possible to retrieve the original structure when several extracted subsets are joined to define the sub-ontology. The set of $\mathcal{SHOIN}(\mathcal{D})$ structural constraints, used to ensure the structurally consistent extraction of axioms, is exactly the same as the one described for the $\mathcal{SHOIN}(\mathcal{D})$ ontology structural consistency preventive resolution approach (c.f. Table 6. $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraints in Chapter 5 Section 2.1).

According to Table 6, for the row *Concept,* the extraction of a concept implies the extraction of dependant axioms of the following subsets:

- sC called set of concepts,
- ≤C the partial orders on the set of concepts sC,
- A function σR:sR→sC2 called role signature,
- A function σA:sA→sC × sT called attribute signature,
- A function ι_C:$sC \rightarrow 2^{sI}$ called concept instantiation,
- A function ε_C:$sC \rightarrow 2^{sC}$ called concept equivalence,
- A function δ_C:$sC \rightarrow 2^{sC}$ called concept disjunction,
- A function -$_C$:$sC \rightarrow 2^{sC}$ called concept complement specification,
- A function $\sqcap_C$:$sC \rightarrow 2^{sC}$ called concept intersection,
- A function $\sqcup_C$:$sC \rightarrow 2^{sC}$ called concept union,

- A function $\rho_{\exists R}$:$sR \rightarrow 2^{sC}$ called role existential restriction,
- A function $\rho_{\forall R}$:$sR \rightarrow 2^{sC}$ called role universal restriction,

For the row *Datatype,* the extraction of a datatype implies the extraction of dependant axioms of the following subsets:

- sT called set of datatypes,
- $\leq_T$ a partial order on the set of datatypes sT,
- A function $\rho_{\exists A}$:$sA \rightarrow 2^{sT}$ called attribute existential restriction,
- A function $\rho_{\forall A}$:$sA \rightarrow 2^{sT}$ called attribute universal restriction,

For the row *Role,* the extraction of a role implies the extraction of dependant axioms of the following subsets:

- sR called *set of roles,*
- $\leq_R$ the partial order on the set of roles sR,
- A function σ_R:$sR \rightarrow sC^2$ called role signature,
- A function ι_R:$sR \rightarrow 2^{sI \times sI}$ called role instantiation,
- A function κ_R:$sR \rightarrow 2^{sKR}$ called role characterization,
- sK_R called set of roles characteristics*,*
- A function ε_R:$sR \rightarrow 2^{sR}$ called role equivalence,
- A function $-_R$:$sR \rightarrow 2^{sR}$ called role inverse specification,
- A function $maxCard_R$:$sR \rightarrow \mathbb{N}$called role maximal cardinality restriction,
- A function $minCard_R$:$sR \rightarrow \mathbb{N}$ called role minimal cardinality restriction,
- A function $\rho_{\exists R}$:$sR \rightarrow 2^{sC}$ called role existential restriction,
- A function $\rho_{\forall R}$:$sR \rightarrow 2^{sC}$ called role universal restriction,
- A function ρ_R:$sR \rightarrow 2^{sI}$ called role value restriction,

For the row *Attribute,* the extraction of an attribute implies the extraction of dependant axioms of the following subsets:

- sA called set of attributes,
- $\leq_A$ *a* partial order on the set of attributes sA,
- A function σ_A:$sA \rightarrow sC \times sT$ called attribute signature,
- A function ι_A:$sA \rightarrow 2^{sI \times sV}$ called attribute instantiation,
- A function κ_A:$sA \rightarrow 2^{sKA}$ called attribute characterization,
- sK_A called set of roles characteristics*;*
- A function ε_A:$sA \rightarrow 2^{sA}$ called attribute equivalence,
- A function $\rho_{\exists A}$:$sA \rightarrow 2^{sT}$ called attribute existential restriction,
- A function $\rho_{\forall A}$:$sA \rightarrow 2^{sT}$ called attribute universal restriction,
- A function ρ_A:$sA \rightarrow 2^{sV}$ called attribute value restriction,

For the row *Instance,* the extraction of an instance implies the extraction of dependant axioms of the following subsets:

- sI called set of instances,
- A function ι_C:$sC \rightarrow 2^{sI}$ called concept instantiation,
- A function ι_R:$sR \rightarrow 2^{sI \times sI}$ called role instantiation,
- A function ι_A:$sA \rightarrow 2^{sI \times sV}$ called attribute instantiation,
- A function ε_I:$sI \rightarrow 2^{sI}$ called instance equivalence,
- A function δ_I:$sI \rightarrow 2^{sI}$ called instance differentiation,
- A function $\sqsubset_I$:$sI \rightarrow 2^{sI}$ called concept enumeration,
- A function ρ_R:$sR \rightarrow 2^{sI}$ called role value restriction,

For the row *Datavalue,* the extraction of a datavalue implies the extraction of dependant axioms of the following subsets:

- sV called set of datavalues,
- A function $\iota_T{:}sT \rightarrow 2^{sV}$ called datatype instantiation,
- A function $\iota_A{:}sA \rightarrow 2^{sI \times sV}$ called attribute instantiation,
- A function $\sqsubset_V{:}sV \rightarrow 2^{sV}$ called datavalue enumeration,
- A function $\rho_A{:}sA \rightarrow 2^{sV}$ called attribute value restriction.

Structural constraints allow **ensuring the structural consistency of ontological element extractions**. However they are not sufficient to define their bounds. For example, according to these constraints, the extraction of a concept implies the extraction of axioms defined by the concept partial order $\leq_C$. If the concept has a many levels of sub-concepts or super-concepts, the extraction of the whole concept subsumption hierarchy is not relevant to **produce a minimal subset** of the ontology. Therefore, the extraction result has to be **limited to the closest surrounding** of the starter elements.

1.3 Extraction Levels

This part defines the **three extraction levels** composing the **view extraction process** proposed in OntoVersionGraph, i.e. the **extraction rule level**, and the **complete and partial extraction levels**.

Extraction rules are defined to guaranty the extraction of minimal subsets $\mathcal{S}_O$', i.e. limited to the closest surrounding of the starter element (c.f. Definition 18).

Definition 18: An **extraction rule** is an algorithm extracting a subset $\mathcal{S}_O{'}=(\Omega_O{'}, \Sigma_O{'}, \Phi_O{'}, E_O)$ of axioms from an ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$, with $\Omega_O{'} \subseteq \Omega_O, \Sigma_O{'} \subseteq \Sigma_O, \Phi_O{'} \subseteq \Phi_O$. It is defined by a starter element ε_i preserving $\mathcal{S}_O$' consistency and ensuring it is a minimal subset.

By **traversing the ontology from the starter element**, the extraction algorithm, defined in an extraction rule, is able to extract a pertinent and consistent material surrounding the starter element. It allows extracting it with a lower semantic loss than a coarse extraction. To produce **minimal subsets**, an extraction rule involves different extractions types of ontological elements, such as concepts, datatypes, roles, attributes, instances or datavalues. These extractions types are differentiated, for each element type, between partial and complete ones (c.f. Definitions 19, 20, 21, 22). A **complete extraction** of an ontological element covers the **whole range of axioms dependant to this element** according to $\mathcal{SHOIN}(\mathcal{D})$ structural constraints, whereas a **partial extraction** only extracts the element in its **strict axiomatic description**, i.e. as an element of the corresponding underlying set and of the functions set that do not involve any other element than itself (like characteristics or cardinality for a role). A **complete extraction uses partial extractions** of the ontological elements, potentially appearing in the axioms dependant to it, in order to **limit the depth of the extraction traversal.**

The three levels considered in a view extraction are:

1. The **extraction rule level** which processes the complete extraction of the starter element,
2. The **complete extraction level** which processes the extraction of the axioms dependant to the starter element and the partial extractions of the other ontological elements appearing in these axioms,
3. The **partial extraction level**, which extract these ontological elements in their strict definition.

Table 10 shows the distribution of complete and partial extractions for each extraction rule on the previous dependency matrix (c.f. Table 6 $\mathcal{SHOIN}(\mathcal{D})$ Structural Constraints in Chapter 5 Section 2.1). The

value *x*, i.e. *dependency[i][j]=x*, still indicates that the extraction of an ontological element related to the row *i* is conditioned by the extraction of the axioms of the subset related to the column j. A red cross indicates that these axioms are extracted within a complete extraction of this element. A grey cross indicates that the other elements composing the axioms are extracted within a partial extraction of this element. The three levels of extractions are formally defined according to this distribution in Section 2.

Extraction Rule Type	sC	sT	sR	sA	sI	sV	sKR	sKA	$\sqsubseteq_C$	$\sqsubseteq_T$	$\sqsubseteq_R$	$\sqsubseteq_I$	σ_R	σ_A	ι_C	ι_T	ι_R	ι_A	κ_R	κ_A	ε_C	ε_R	ε_A	ε_I	δ_C	δ_I	$\_\_C$	$\_\_R$	$maxCard_R$	$minCard_R$	$\sqcap_C$	$\sqcup_C$	$\sqcup_I$	$\sqcup_V$	$\rho_{\exists R}$	$\rho_{\forall R}$	ρ_R	$\rho_{\exists A}$	$\rho_{\forall A}$	ρ_A
Concept	x								xx				x	x	x						xx				xx		xx				xx	xx			x	x				
Datatype		x								xx				x		x																			x	x			
Role	x		x		x		x				xx		x				x		x			xx						xx	x	x					x	x	x			
Attribute	x	x		x	x	x		x				xx		x				x		x			xx												x	x	x			
Instance					x										x		x	x						xx		xx							x				x				
Datavalue						x										x		x																xx						x

Table 10. Complete and Partial Extractions Distribution according to Structural Constraints and Extraction Rule Types

1.4 Discussion

This part explains how the OntoVersionGraph methodology **evolution and versioning features** can be used to manage such views.

To evolve and version the sub-ontology produced by such a view, a **global log of changes** has to be built. The global log of changes of the sub-ontology is a **subset of the global log of changes of the source ontology**. To built it, the set of axioms composing the view are required. The log of changes of the sub-ontology can then be generated **extracting each addition change from the global log of changes corresponding to an axiom of the sub-ontology**. The sub-ontology can then be evolved and versioned in the OntoVersionGraph change management system like any other $\mathcal{SHOIN}(\mathcal{D})$ ontology.

In addition, the extraction of a sub-ontology has to be **represented in the global log of changes** of the source ontology, to **localize the extraction according to the evolution timeline** of the ontology. The dating of the extraction is essential in case of a **change rollback**, i.e. a return to a previous version of the ontology. For example, let us consider having a version $\mathcal{S}_{O^{n+i}}$ of an ontology. The extraction of a view has been done on a version $\mathcal{S}_{O^n}$ of the ontology. If we want to rollback to a version $\mathcal{S}_{O^{n-1}}$ then we expect to generate the previous version of the ontology without the view. However, the extraction of a sub-ontology is **not a change as strictly defined** in our model, because it does not modify the ontology. But as it is part of the ontology change management methodology, it is convenient to **represent it as a change in the global log of changes** of the ontology and to localize it relatively to the others changes occurred. It can then be represented as a **complex change containing all the changes represented in the sub-ontology global log of changes**.

2 View Extraction Formalization

This section formalizes the process of sub-ontology extraction from a $\mathcal{SHOIN}(\mathcal{D})$ ontology. The **three extraction levels of a view extraction**, controlled by structural constraints and limited by the use of **complete** and **partial extractions**, allow producing **minimal subsets** in the view extraction result. This section defines the entire **view extraction process** by formalizing these three extraction levels. A first part defines the global $\mathcal{SHOIN}(\mathcal{D})$ view extraction. A second part defines the **extraction rules** corresponding to the first level. A third part defines the **complete extractions** related to the second level, which are used by extraction rules. A fourth part defines the **partial extractions** of the third level, which are used by complete extractions.

In order to ease the understanding of the following, it has to be noted that the extraction of a view uses six kinds of **extraction rules**, i.e. **starter concept** (c.f. Definition 20), **starter datatype** (c.f. Definition 21), **starter role** (c.f. Definition 22), **starter attribute** (c.f. Definition 23), **starter instance** (c.f. Definition 24) and **starter datavalue** (c.f. Definition 25). In addition, each starter extraction rule makes use of complete extractions, which make use of partial extractions (c.f. Fig. 18).

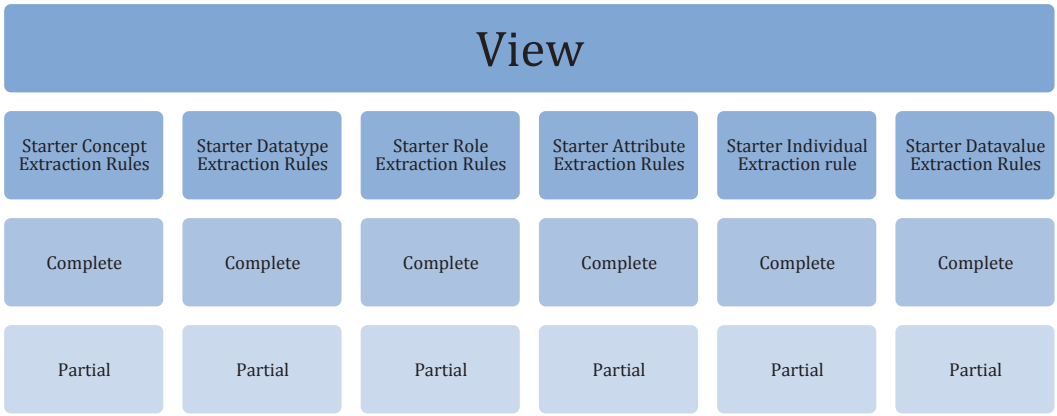


Figure 18. Hierarchy of Extraction rules defined for the Extraction of a $\mathcal{SHOIN}(\mathcal{D})$ View

2.1 $\mathcal{SHOIN}(\mathcal{D})$ View Extraction

This part formalizes the global $\mathcal{SHOIN}(\mathcal{D})$ view extraction process using the six kinds of starter extraction rules (c.f. Definition 19).

Definition 19: The **extraction process** of a view V defined by a view definition $V(\mathcal{S}_O)$ on an ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$, consists in joining the six extraction rule results $\mathcal{S}_O{}^C$, $\mathcal{S}_O{}^R$, $\mathcal{S}_O{}^A$, $\mathcal{S}_O{}^I$, $\mathcal{S}_O{}^T$ and $\mathcal{S}_O{}^V$ applied an ontology $\mathcal{S}_O$. The **view result** is then defined by $V_{\mathcal{S}_O}=\{\mathcal{S}_O{}^C, \mathcal{S}_O{}^T, \mathcal{S}_O{}^R, \mathcal{S}_O{}^A, \mathcal{S}_O{}^I, \mathcal{S}_O{}^V\}$ such that:

- $\mathcal{S}_O{}^C = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{C_i}$, for each starter concept $C_i \in sC \subseteq \Omega_O$, such as $\mathcal{S}_O{}^{C_i}=E(C_i)$ is the result of the starter extraction rule based on C_i.
- $\mathcal{S}_O{}^T = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{T_i}$, for each starter datatype $T_i \in sT \subseteq \Omega_O$, such as $\mathcal{S}_O{}^{T_i}=E(T_i)$ is the result of the starter extraction rule based on T_i.
- $\mathcal{S}_O{}^R = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{R_i}$, for each starter role $R_i \in sR \subseteq \Omega_O$, such as $\mathcal{S}_O{}^{R_i}=E(R_i)$ is the result of the starter extraction rule based on R_i.
- $\mathcal{S}_O{}^A = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{A_i}$, for each starter attribute $A_i \in sA \subseteq \Omega_O$, such as $\mathcal{S}_O{}^{A_i}=E(A_i)$ is the result of the starter extraction rule based on A_i.

- $\mathcal{S}_O{}^I = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{I_i}$, for each starter instance $I_i \in sI, \subseteq \Omega_O$ such as $\mathcal{S}_O{}^{I_i} = E(I_i)$ is the result of the starter extraction rule based on I_i.
- $\mathcal{S}_O{}^V = \bigcup_{i=1}^{n} \mathcal{S}_O{}^{V_i}$, for each starter datavalue $V_i \in sV \subseteq \Omega_O$, such as $\mathcal{S}_O{}^{V_i} = E(V_i)$ is the result of the starter extraction rule based on V_i.

2.2 Starter Extraction rules

This part formalizes the six starter extraction rule types used in the view extraction process defined above, i.e. **starter concept**, **starter datatype**, **starter role**, **starter attribute**, **starter instance** and **starter datavalue** extraction rules.

Definition 20: The **starter concept extraction rule** is an extraction that has a concept as input starter element. Given a concept $D_i \in sC$ the extraction rule resulting from a starter element D_i is a sub-ontology $\mathcal{S}_O{}^{Di}$ defined by $\mathcal{S}_O{}^{Di} = E(D_i) = \{\theta_{cDi}\}$ with θ_{cDi} the complete extraction of the concept D_i.

Definition 21: The **starter datatype extraction rule** defines an extraction that has a datatype as input starter element. Given a datatype $T_i \in sT$ the extraction rule resulting from a starter element T_i is a sub-ontology $\mathcal{S}_O{}^{Ti}$ defined by $\mathcal{S}_O{}^{Ti} = E(T_i) = \{\theta_{cTi}\}$ with θ_{cTi} the complete extraction of the datatype T_i.

Definition 22: The **starter role extraction** rule defines an extraction that has a role as input starter element. Given a role $R_i \in sR$, the extraction rule resulting from a starter R_i as a starter is a sub-ontology $\mathcal{S}_O{}^{Ri}$ defined by $\mathcal{S}_O{}^{Ri} = E(R_i) = \{\theta_{cRi}\}$ with θ_{cRi} the complete extraction of the role R_i.

Definition 23: The **starter attribute extraction** rule defines an extraction that has an attribute as input starter element. Given an attribute $A_i \in sA$, the extraction of A_i as a starter is a sub-ontology $\mathcal{S}_O{}^{Ai}$ as defined by $\mathcal{S}_O{}^{Ai} = E(A_i) = \{\theta_{cAi}\}$ with θ_{cAi} the complete extraction of the attribute A_i.

Definition 24: The **starter instance extraction rule** defines an extraction that has an instance as input starter element. Given an instance $I_i \in sI$, the extraction of I_i as a starter element is a sub-ontology $\mathcal{S}_O{}^{Ii}$ defined by $\mathcal{S}_O{}^{Ii} = E(I_i) = \{\theta_{cIi}\}$ with θ_{cIi} the complete extraction of the instance I_i.

Definition 25: The **starter datavalue extraction rule** defines an extraction that has a datavalue as input starter element. Given a datavalue $V_i \in sV$, the extraction of V_i as a starter element is a sub-ontology $\mathcal{S}_O{}^{Vi}$ defined by $\mathcal{S}_O{}^{Vi} = E(V_i) = \{\theta_{cVi}\}$ with θ_{cV} the complete extraction of the datavalue V_i.

.

2.3 Complete Extraction rules

This part formalizes the **six complete extraction rules** used by the six starter extraction rules, i.e. **concept, datatype, role, attribute, instance** and **datavalue** complete extraction rules. They are defined as sets of axioms. It has to be noted that all the complete extraction rules follow the $\mathcal{SHOIN}(\mathcal{D})$ structural constraints. More details are given regarding the selection of the constraints for each rule.

Definition 26: The **complete concept extraction rule** is defined as follows. Given a concept $D_i \in sC$, the complete extraction of D_i noted $\theta_{cDi} = \{\theta_{pDi}, (\theta_{pDsup}, \leq^{\theta}_{Disup}), (\theta_{pDsub}, \leq^{\theta}_{\mathrm{Disub}}), (\varepsilon_C^{Di}, \theta_{pD\varepsilon_C}), (-_C^{Di}, \theta_{pD-_C}), (\delta_C^{Di}, \theta_{pD\delta_C}), (\sqcap_C^{Di}, \theta_{pD\sqcap_C}), (\sqcup_C^{Di}, \theta_{pD\sqcup_C}), \sigma_{\mathrm{R}}^{\mathrm{Di}}, \theta_{pR_{\sigma_{Ri}}}, \theta_{pD_{\sigma_{Ri}}}, \sigma_A^{Di}, \theta_{pD_{\sigma_{Ai}}}, \theta_{pT_{\sigma_{Ai}}}, \rho_{\forall R}^{Di}, \theta_{pR\rho_{\forall R}}, \rho_{\exists R}^{Di}, \theta_{pR\rho_{\exists R}}\}$ in the extraction result is defined by:

- The partial extraction of D_i, noted θ_{pDi}=$\{ D_i^{\theta}, \leq_{Ditop}^{\theta} \}$ as defined by θ_{pDi}=$\{ TopConcept, D_i \}$ and $\leq_{Ditop}^{\theta}$ =$\{(TopConcept, D_i)\}$;
- The extraction of all direct super-concept partial orders of D_i, noted $\leq_{Disup}^{\theta}$, as defined by $\leq_{Disup}^{\theta}$=$\{(D_{Sup_1}, D_i), \ldots, (D_{Sup_n}, D_i)\}$ with $D_{Sup_j} \in sC$ and $j \in \mathbb{N}$;
- The partial extraction of all direct super-concepts of D_i *in* $\leq_{Disup}^{\theta}$, noted θ_{pDisup}, as defined by θ_{pDsup}= $\bigcup_{i=1}^{n} \theta_{pD_{Sup_j}}$ and $j \in \mathbb{N}$;
- The extraction of all direct sub-concept partial orders of D_i, *noted* $\leq_{Disub}^{\theta}$, as defined by $\leq_{Disub}^{\theta}$=$\{(D_{sub_1}$, $D_i), \ldots, (D_{sub_n}, D_i)\}$ *with* $D_{sub_j} \in sC$ and $j \in \mathbb{N}$;
- The partial extraction of all direct sub-concepts of D_i *in* $\leq_{Disub}^{\theta}$, *noted* θ_{pDiub}, as defined by θ_{pDsub}= $\bigcup_{i=1}^{n} \theta_{pD_{sub_j}}$ and $j \in \mathbb{N}$;
- The extraction of all the concept equivalences of ε_C where D_i is defined as equivalent to another concept, noted ε_C^{Di}, *as defined by* ε_C^{Di} =$\{(D_i, D_1), \ldots, (D_i, D_n), (D_{n+1}, D_i), \ldots, (D_{n+m}, D_i) \}$, *with* $D_j \subseteq sC$ and $j \in \mathbb{N}$;
- The partial concept extraction of all the concepts equivalent to D_i *in* $\varepsilon(D_i)$, noted $\theta_{pD\varepsilon_C}$ as defined by $\theta_{pD\varepsilon_C}$=$\bigcup_{i=1}^{n} \theta_{pD_{\varepsilon_{Cj}}}$ and $j \in \mathbb{N}$;
- The extraction of all the concept complements of $-_C$, where D_i is defined as a complement of another concept, noted $-_C^{Di}$, *as defined by* $-_C^{Di}$ =$\{(D_i, D_1), \ldots, (D_i, D_n), (D_{n+1}, D_i), \ldots, (D_{n+m}, D_i) \}$, *with* $D_j \subseteq sC$ and $j \in \mathbb{N}$.
- The partial concept extraction of all the concepts complementary to D_i in $-_C^{Di}$, noted θ_{pD-_C}, as defined by θ_{pD-_C}=$\bigcup_{i=1}^{n} \theta_{pD_{-_{Cj}}}$ and $j \in \mathbb{N}$;
- The extraction of all the concept disjunctions of δ_C where D_i is disjoint with another concept, noted δ_C^{Di}, as defined by δ_C^{Di} =$\{(D_i, D_1), \ldots, (D_i, D_n), (D_{n+1}, D_i), \ldots, (D_{n+m}, D_i) \}$, with $D_j \subseteq sC$ and $j \in \mathbb{N}$.
- The partial concept extraction of all the concepts disjoint with D_i in δ_C^{Di}, noted $\theta_{pD\delta_C}$, as defined by $\theta_{pD\delta_C}$= $\bigcup_{i=1}^{n} \theta_{pD_{\delta_{Cj}}}$ and $j \in \mathbb{N}$;
- The extraction of all the concept intersections of $\sqcap_C$ where D_i is intersected with another concept, noted $\sqcap_C^{Di}$, as defined by $\sqcap_C^{Di}$=$\{(D_i, D_1), \ldots, (D_i, D_n), (D_{n+1},), \ldots, (D_{n+m}, D_i) \}$, with $D_j \subseteq sC$ and $j \in \mathbb{N}$.
- The partial concept extraction of all the concepts intersected with D_i in $\sqcap_C^{Di}$, noted $\theta_{pD\sqcap_C}$, as defined by $\theta_{pD\sqcap_C}$=$\bigcup_{i=1}^{n} \theta_{pD_{\sqcap_{Cj}}}$ and $j \in \mathbb{N}$;
- The extraction of all the concept unions of $\sqcup_C$ where D_i is joined with another concept, noted $\sqcup_C^{Di}$, as defined by $\sqcup_C^{Di}$=$\{(D_i, D_1), \ldots, (D_i, D_n), (D_{n+1}, D_i), \ldots, (D_{n+m}, D_i) \}$, with $D_j \subseteq sC$ and $j \in \mathbb{N}$.
- The partial concept extraction of all the concepts joined with D_i *in* $\sqcup_C^{Di}$, noted $\theta_{pD\sqcup_C}$, is defined by $\theta_{pD\sqcup_C}$ =$\bigcup_{i=1}^{n} \theta_{pD_{\sqcup_{Cj}}}$ and $j \in \mathbb{N}$;
- The extraction of all the role signatures *of* σ_R where D_i is defined as domain or range, noted σ_R^{Di} defined by σ_R^{Di} =$\{(R_1, (D_i, D_1)), \ldots, (R_n, (D_i, D_m)), (R_{n+1}, (D_{m+1}, D_i)), \ldots, (R_{n+o}, (D_{n+p}, D_i)) \}$ with $R_j \in sR$, $D_k \in sC$, and $j, k \in \mathbb{N}$;
- The partial extraction of all roles defined in $\sigma_R(R)$, noted $\theta_{pR_{\sigma_R}}$, as defined by $\theta_{pR_{\sigma_R}}$= $\bigcup_{i=1}^{n} \theta_{pR_{\sigma_{R_j}}}$ and $j \in \mathbb{N}$;
- The partial extraction of all concepts D_j, different from D_i, defined as domain or range in σ_R^{Di}, noted $\theta_{pD_{\sigma_R}}$, as defined by $\theta_{pD_{\sigma_R}}$= $\bigcup_{i=1}^{n} \theta_{pD_{\sigma_{R_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all the attribute signatures of σ_A where D_i is defined as domain, noted σ_A^{Di}, as defined by σ_A^{Di} =$\{(A_1, (D_i, T_1)), \ldots, (A_n, (D_i, T_m)) \}$ with $A_j \in sA$, $T_k \in sT$, and $j, k \in \mathbb{N}$.
- The partial extraction of all attributes defined in σ_A^{Di}, noted $\theta_{pA_{\sigma_A}}$, as defined by $\theta_{pA_{\sigma_A}}$= $\bigcup_{i=1}^{n} \theta_{pA_{\sigma_{A_j}}}$ and $j \in \mathbb{N}$;

- The partial extraction of all datatypes defined as range in σ_A^{Di}, noted $\theta_{pT_{\sigma_A}}$, as defined by $\theta_{pT_{\sigma_A}} = \bigcup_{i=1}^{n} \theta_{pT_{\sigma_{A_j}}}$ and $j \in \mathbb{N}$.
- The extraction of all universal restrictions of $\rho_{\forall R}$where D_i is defined as range, noted $\rho_{\forall R}^{Di}$, as defined by $\rho_{\forall R}^{Di}$={$(R_1,(D_i))...(R_n, (D_i))$} with $R_j \in sR$ and $j \in \mathbb{N}$;
- The partial extraction of all roles defined as role in $\rho_{\forall R}^{Di}$, noted $\theta_{pR\rho_{\forall R}}$, is defined by $\theta_{pR\rho_{\forall R}} = \bigcup_{i=1}^{n} \theta_{pR_{\rho_{\forall R_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all existential restrictions of $\rho_{\exists R}$ where D_i is defined as range, noted $\rho_{\exists R}^{Di}$, as defined by $\rho_{\exists R}^{Di}$={$(R_1,(D_i))..., (R_n, (D_i))$} with $R_j \in sR$ and $j \in \mathbb{N}$;
- The partial extraction of all roles defined as role in $\rho_{\exists R}^{Di}$, noted $\theta_{pR\rho_{\exists R}}$, is defined by $\theta_{pR\rho_{\exists R}} = \bigcup_{i=1}^{n} \theta_{pR_{\rho_{\exists R}j}}$ and $j \in \mathbb{N}$;

Definition 27: The **complete datatype extraction rule** is defined as follows. Given a data type $T_i \in sT$, the complete extraction of T_i, noted θ_{cTi}={θ_{pTi}, ($\leq_{Tisup}^{\theta}$, θ_{pTsup}), ($\leq_{Tisub}^{\theta}$, θ_{pTsub}), $\rho_{\forall A}^{Ti}$, $\theta_{pA\rho_{\forall A}}$, $\rho_{\exists A}^{Ti}$, $\theta_{pA\rho_{\exists A}}$}, in the extraction result, is defined by :

- The partial extraction of T_i noted θ_{pTi}={T_i^{θ}, $\leq_{Titop}^{\theta}$};
- The extraction of all direct super-type partial orders of T_i, noted $\leq_{TiSup}^{\theta}$, is defined by $\leq_{TiSup}^{\theta}$={(T_{sup1},T_i), ...,(T_{supn}, T_i)} with $T_{supj} \in sT$ and $j \in \mathbb{N}$;
- The partial extraction of all the direct super-types of T_i in $\leq_{TiSup}^{\theta}$, noted θ_{pTsup}, is defined by θ_{pTsup}= $\bigcup_{i=1}^{n} \theta_{pT_{Sup_j}}$ and $j \in \mathbb{N}$;
- The partial order extraction of all direct sub-types of T_i is defined by $\leq_{Tisub}^{\theta}$={(T_i, T_{sub1}), ..., (T_i, T_{subm})} with $T_{subj} \in sT$ and $j \in \mathbb{N}$;
- The partial extraction of all the direct sub-types of T_i in $\leq_{Tisub}^{\theta}$, noted θ_{pTsub}, is defined by θ_{pTsub}= $\bigcup_{i=1}^{n} \theta_{pT_{sub_j}}$ and $j \in \mathbb{N}$;
- The extraction of all attribute universal restrictions of $\rho_{\forall A}$where T_i is defined as range, noted $\rho_{\forall A}^{Ti}$, as defined by $\rho_{\forall A}^{Ti}$={$(A_1,(T_i))...(R_n, (T_i))$} with $A_j \in sA$ and $j \in \mathbb{N}$;
- The partial extraction of all attributes defined as attribute in $\rho_{\forall A}^{Ti}$, noted $\theta_{pA\rho_{\forall A}}$, is defined by $\theta_{pA\rho_{\forall A}} = \bigcup_{i=1}^{n} \theta_{pA_{\rho_{\forall A_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all attribute existential restrictions of $\rho_{\exists A}$ where T_i is defined as range, noted $\rho_{\exists A}^{Ti}$, as defined by $\rho_{\exists A}^{Ti}$={$(A_1,(T_i))..., (A_n, (T_i))$} with $A_j \in sA$ and $j \in \mathbb{N}$;
- The partial extraction of all atributes defined as attribute in $\rho_{\exists A}^{Ti}$, noted $\theta_{pA\rho_{\exists A}}$, is defined by $\theta_{pA\rho_{\exists A}} = \bigcup_{i=1}^{n} \theta_{pA_{\rho_{\exists A}j}}$ and $j \in \mathbb{N}$;

Definition 28: The **complete role extraction rule** is defined as follows. Given a role $R_i \in sR$, a complete extraction of R_i, noted θ_{cRi}={ θ_{pRi}, ε_R^{Ri}, $\theta_{pR\varepsilon_R}$, $\leq_{Risup}^{\theta}$, θ_{pRsup}, $\leq_{Risub}^{\theta}$, θ_{pRsub}, $-_R^{Ri}$, θ_{pR-_R}, σ_R^{Ri}, $\theta_{pC\sigma_R}$, $\rho_{\forall R}^{Ri}$, $\theta_{pC\rho_{\forall R}}$, $\rho_{\exists R}^{Ri}$, $\theta_{pC\rho_{\exists R}}$, ρ_R^{Ri}, $\theta_{pI\rho_R}$} in the extraction result, is a set containing:

- The partial extraction of R_i, noted θ_{pRi}, as defined by θ_{pRi} ={ R_i^{θ}, $\leq_{Ritop}^{\theta}$, $\kappa_R(Ri)$, $sK_R(R_i)$, $maxCard_R(R_i)$, $minCard_R(R_i)$ };
- The extraction of all role equivalences of $\varepsilon_R$where R_i is defined as equivalent to another role, noted ε_R^{Ri}, as defined by $\varepsilon_R^{Ri} = \{(Ri, R_1), \dots, (Ri, R_n)\}$, with $R_{ij} \in sR$ and $j \in \mathbb{N}$;
- The partial extraction of all roles equivalent to R_i in ε_R^{Ri}, noted $\theta_{pRi\varepsilon_R}$, as defined by $\theta_{pR\varepsilon_R}$= $\bigcup_{i=1}^{n} \theta_{pR_{\varepsilon_{Rj}}}$ and $j, n \in \mathbb{N}$;
- The extraction of all direct super role partial orders of R_i, noted $\leq_{Risup}^{\theta}$, as defined by $\leq_{Risup}^{\theta}$={(R_{sup1}, R_i),

…,(R_{supn}, R_i)} with $R_{supj} \in sR$ and $j \in \mathbb{N}$;

- The partial extraction of all direct super-roles of R_i in $\leq^{\theta}_{Risup}$, noted θ_{pRsup}, as defined by $\theta_{pRsup} = \bigcup_{i=1}^{n} \theta_{pR_{sup_j}}$ and $j \in \mathbb{N}$;
- The extraction of all direct sub-role partial orders of R_i, noted $\leq^{\theta}_{Risub}$, as defined by $\leq^{\theta}_{Risub}$={(R_i, R_{sub1}), …, (R_i, R_{subm})} with $R_{subi} \in sR$ and $j \in \mathbb{N}$
- The partial extraction of all direct sub-roles of R_i in $\leq^{\theta}_{Risub}$, noted θ_{pRsub}, as defined by $\theta_{pRsub} = \bigcup_{i=1}^{n} \theta_{pR_{Sub_i}}$ and $n \in \mathbb{N}$;
- The extraction of all role inversions of $-_R$, where R_i is defined as inverse of another role, noted $-_R^{Ri}$, as defined by $-_R^{Ri}$={(R_i,R_{-1})…(R_i,R_{-n})} with $R_{-j} \in sR$ and $j \in \mathbb{N}$
- The partial extraction of all inverse roles of R_i in $-_R^{Ri}$, noted θ_{pR-R}, as defined by $\theta_{pR-R} = \bigcup_{i=1}^{n} \theta_{pR-R_j}$ and $j \in \mathbb{N}$;
- The extraction of all the role signatures of σ_R where R_i is defined as role, noted σ_R^{Ri}, as defined by σ_R^{Ri}={(R_i,(C_1, C_1))… (R_i,(C_n, C_m))} with $C_j \in sC$, and $j \in \mathbb{N}$;
- The partial extraction of all concepts in the domain or range of each signature of R_i in σ_R^{Ri}, noted $\theta_{pC\sigma_R}$, as defined by $\theta_{pC\sigma_R} = \bigcup_{i=1}^{n} \theta_{pC_{\sigma_{R_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all universal restrictions of $\rho_{\forall R}$ where R_i is defined as role, noted $\rho_{\forall R}^{Ri}$, as defined by $\rho_{\forall R}^{Ri}$={(R_i,($D_{\rho\forall 1}$))…(R_i, ($D_{\rho\forall n}$)} with $D_{\rho\forall j} \in sC$ and $j \in \mathbb{N}$;
- The partial extraction of all concepts defined as range of each universal restriction of R_i in $\rho_{\forall R}^{Ri}$, noted $\theta_{pC\rho_{\forall R}}$, is defined by $\theta_{pC\rho_{\forall R}} = \bigcup_{i=1}^{n} \theta_{pC_{\rho_{\forall R_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all existential restrictions of $\rho_{\exists R}$ where R_i is defined as role, noted $\rho_{\exists R}^{Ri}$, as defined by $\rho_{\exists R}^{Ri}$={(R_i, ($D_{\rho\exists 1}$)…(R_i, ($D_{\rho\exists n}$)} with $D_{\rho\exists j} \in sC$ and $j \in \mathbb{N}$;
- The partial extraction of all concepts in the range of each existential restriction of R_i in $\rho_{\exists R}^{Ri}$, noted $\theta_{pC\rho_{\exists R}}$, is defined by $\theta_{pC\rho_{\exists R}} = \bigcup_{i=1}^{n} \theta_{pC_j}$ and $j \in \mathbb{N}$;
- The extraction of all value restrictions of $\rho_R(R)$ where R_i is defined as role, noted ρ_R^{Ri} as defined by ρ_R^{Ri}={(R_i,($I_{\rho 1}$)…(R_i, ($I_{\rho n}$)} with $I_{\rho j} \in sI$ and $j \in \mathbb{N}$;
- The partial extraction of all instances in the range of each value restriction of R_i in ρ_R^{Ri}, noted $\theta_{pI\rho_R}$, is defined by $\theta_{pI\rho_R} = \bigcup_{i=1}^{n} \theta_{pI_{\rho_{R_j}}}$ and $j \in \mathbb{N}$.

Definition 29: The **complete attribute extraction rule** is defined as follows. Given an attribute $A_i \in sA$, a complete extraction of A_i, noted θ_{cAi} ={θ_{pAi}, ε_A^{Ai}, $\theta_{pA_{\varepsilon_A}}$, $\leq^{\theta}_{Ai_{sup}}$, $\theta_{pA_{sup}}$, $\leq^{\theta}_{Ai_{sub}}$, $\theta_{pAi_{sub}}$, σ_A^{Ai}, $\theta_{pC_{\sigma_A}}$, $\theta_{pT_{\sigma_A}}$, $\rho_{\forall A}^{Ai}$, $\theta_{pT\rho_{\forall A}}$, $\rho_{\exists A}^{Ai}$, $\theta_{pT\rho_{\exists A}}$, ρ_A^{Ai}, $\theta_{pV_{\rho_A}}$ }, is a set containing :

- The partial extraction of A_i, noted θ_{pAi}, as defined by θ_{pAi}= {A_i^{θ}, $\leq^{\theta}_{Ai_{top}}$, $\kappa_A(Ai)$, sK_A^{θ}};
- The extraction of all attribute equivalences of $\varepsilon_A(A)$ where A_i is defined as an attribute equivalent to another attribute, noted ε_A^{Ai}, as defined by $\varepsilon_A^{Ai} = \{(Ai, A_{\varepsilon 1}), \ldots, (Ai, A_{\varepsilon n})\}$, with $A_{\varepsilon j} \in sA$ and $j \in \mathbb{N}$;
- The partial extraction of all attributes equivalent to A_i *in* ε_A^A, noted $\theta_{pA_{\varepsilon_A}}$, as defined by $\theta_{pA_{\varepsilon_A}} = \bigcup_{i=1}^{n} \theta_{pA_{\varepsilon_{Aj}}}$ and $j \in \mathbb{N}$;
- The extraction of all direct super-attribute partial orders of A_i, noted $\leq^{\theta}_{Ai_{sup}}$, as defined by $\leq^{\theta}_{Ai_{sup}}$={(A_{sup1}, A_i), …, (A_{supn}, A_i)} *with* $A_{supj} \in sA$ and $j \in \mathbb{N}$;
- The partial extraction of all direct super-attributes of A_i *in* $\leq^{\theta}_{A_{sup}}$, noted $\theta_{pA_{sup}}$, as defined by $\theta_{pA_{sup}} = \bigcup_{i=1}^{n} \theta_{pA_{sup_j}}$ *and* $j \in \mathbb{N}$;
- The extraction of all direct sub-attribute partial orders of A_i, noted $\leq^{\theta}_{Ai_{sub}}$, as defined by $\leq^{\theta}_{Ai_{sub}}$={(A_i, A_{sub_1}), …, (A_i, A_{sub_n})} with $A_{sub_j} \in sA$ and $j \in \mathbb{N}$;

- The partial extraction of all direct sub-attributes of A_i *in* $\leq^{\theta}_{A\,sub}$, noted $\theta_{pA_{sub}}$, as defined by $\theta_{pA_{sub}} = \bigcup_{i=1}^{n} \theta_{pA_{sub_j}}$ *and* $j \in \mathbb{N}$;
- The extraction of all the attribute signatures of σ_A where A_i, is defined as an attribute, noted σ_A^{Ai}, as defined by σ_A^{Ai}=$\{(A_i, (C_1, T_1))..., (A_i, (C_n, T_m))\}$ with $C_j \in sC$, $T_k \in sT$, and $j, k \in \mathbb{N}$;
- The partial extraction of all concepts in the domain of each attribute signature of σ_A^A, noted $\theta_{pC_{\sigma_A}}$, as defined by $\theta_{pC_{\sigma_A}} = \bigcup_{i=1}^{n} \theta_{pC_{\sigma_{A_j}}}$ and $j \in \mathbb{N}$;
- The partial extraction of all datatypes in the range of each attribute signature of σ_A^A, noted $\theta_{pT_{\sigma_A}}$, as defined by $\theta_{pT_{\sigma_A}} = \bigcup_{i=1}^{n} \theta_{pT_{\sigma_{A_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all universal restrictions of $\rho_{\forall A}$ where A_i is defined as an attribute, noted $\rho_{\forall A}^{Ai}$, as defined by $\rho_{\forall A}^{Ai}$=$\{(A_i, (T_{\rho\forall 1}))...(A_i, (T_{\rho\forall n}))\}$ with $T_{\rho\forall j} \in sT$ and $j \in \mathbb{N}$;
- The partial extraction of all datatypes in the range of each universal restriction of A_i in $\rho_{\forall A}^{Ai}$, noted $\theta_{pT\rho_{\forall A}}$, is defined by $\theta_{pT\rho_{\forall A}} = \bigcup_{i=1}^{n} \theta_{pT_{\rho_{\forall A_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all existential restrictions of $\rho_{\exists A}$ where A_i is defined as an attribute, noted $\rho_{\exists A}^{Ai}$, as defined by $\rho_{\exists A}^{Ai}$=$\{(A_i, (T_{\rho\exists 1}))...(A_i, (T_{\rho\exists n}))\}$ with $T_{\rho\exists j} \in sT$ and $j \in \mathbb{N}$;
- The partial extraction of all datatypes in the range of each existential restriction of A *in* $\rho_{\exists A}^{Ai}$, noted $\theta_{pT\rho_{\exists A}}$, is defined by $\theta_{pT\rho_{\exists A}} = \bigcup_{i=1}^{n} \theta_{pT_{\rho_{\exists A_j}}}$ *and* $j \in \mathbb{N}$;
- The extraction of all value restrictions of $\rho_A$where A_i is defined as an attribute, noted ρ_A^{Ai}, as defined by ρ_A^{Ai}=$(A_i, (V_{\rho 1}))...(A_i, (V_{\rho n}))\}$ with $V_{\rho j} \in sV$ and $j \in \mathbb{N}$;
- The partial extraction of all datavalues in the range of each value restriction of ρ_A^{Ai}, noted $\theta_{pV_{\rho_A}}$, as defined by $\theta_{pV_{\rho_A}} = \bigcup_{i=1}^{n} \theta_{pV_{\rho_{A_j}}}$ and $j \in \mathbb{N}$;

Definition 30: The **complete instance extraction rule** is defined as follows. Given an instance $I_i \in sI$, the complete extraction of I_i as a starter element is a set, noted θ_{cIi}, *as defined by* θ_{cIi}= $\{\theta_{pIi}, \iota_C^{Ii}, \theta_{pD\iota_C}, \iota_R^{Ii}, \theta_{pR_{\iota_R}}, \theta_{pI_{\iota_R}}, \iota_A^{Ii}, \theta_{pA_{\iota_A}}, \theta_{pV_{\iota_A}}, \varepsilon_I^{Ii}, \theta_{pI_{\varepsilon_I}}, \delta_I^I i, \theta_{pI_{\delta_I}}, \sqcup_I^{Ii}, \theta_{pI_{\sqcup I}}, \rho_R^{Ii}, \theta_{pR\rho_R}\}$ *with:*

- The partial extraction of I_i, noted θ_{pIi}, as defined by θ_{pIi}=$\{I_i^{\theta}\}$;
- The extraction of all instance equivalences of ε_I where I_i is defined as an instance equivalent to another instance, noted ε_I^I, as defined by $\varepsilon_I^{Ii} = \{(Ii, I_{\varepsilon 1}), \dots, (Ii, I_{\varepsilon n})\}$, with $I_{\varepsilon j} \in sI$ and $j \in \mathbb{N}$;
- The partial extraction of all instances equivalent to I_i in each instance equivalence of ε_I^{Ii}, noted $\theta_{pI_{\varepsilon_I}}$, as defined by $\theta_{pI_{\varepsilon_I}}$= $\bigcup_{i=1}^{n} \theta_{pI_{\varepsilon_{I_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all instance disjunctions of δ_I where I_i is defined as an instance different from another instance, noted δ_I^{Ii}, as defined by δ_I^{Ii}=$\{(Ii, I), \dots, (Ii, I_{\delta n})\}$, with $I_{\delta j} \in sI$ and $j \in \mathbb{N}$;
- The partial extraction of all instances disjoint with I_i in each instance disjunction of δ_I^{Ii}, noted $\theta_{pI_{\delta_I}}$, as defined by $\theta_{pI_{\delta_I}}$= $\bigcup_{i=1}^{n} \theta_{pI_{\delta_{I_i}}}$ and $j \in \mathbb{N}$.
- The extraction of all concept instantiations of ι_C where I_i is defined as an instance, noted ι_C^{Ii}, as defined by $\iota_C^{Ii} = \{(D_1(Ii)), \dots, (D_n(Ii))\}$ with $D_j \in sC$ and $j \in \mathbb{N}$;
- The partial extraction of all concepts instantiated by I_i in each concept instantiation of ι_C^{Ii}, noted $\theta_{pD_{\iota_C}}$, is defined by $\theta_{pD_{\iota_C}}$= $\bigcup_{i=1}^{n} \theta_{pD_{\iota_{C_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all role instantiations of ι_R where I_i is defined as domain or range, noted ι_R^{Ii}, as defined by $\iota_R^{Ii} = \left\{\left(R_1, (Ii, I_1)\right) \dots \left(R_n, (Ii, I_m)\right) \dots \left(R_{n+1}, (I_{m+1}, Ii)\right) \dots \left(R_{n+o}, \left(I_{m+p}, Ii\right)\right)\right\}$ with $R_j \in sR$, $I_k \in sI$ and $j, k \in \mathbb{N}$,
- The partial extraction of all roles in each role instantiation of ι_R^{Ii} , noted $\theta_{pR_{\iota_R}}$, as defined by $\theta_{pR_{\iota_R}}$= $\bigcup_{i=1}^{n} \theta_{pR_{\iota_{R_j}}}$ and $j \in \mathbb{N}$;

- The partial extraction of all instances different from I_i in the domain or range of each role instantiation of ι_R^{Ii}, noted $\theta_{pI_{\iota_R}}$, as defined by $\theta_{pI_{\iota_R}} = \bigcup_{i=1}^{n} \theta_{pI_{\iota_{R_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all attribute instantiations of ι_A where I_i defined as domain, noted ι_A^{Ii}, as defined by $\iota_A^{Ii} = \{(A1, (I, V_1)) \dots (An, (I, V_m))\}$ with $A_j \in sA$, $V_k \in sV$ and $j, k \in \mathbb{N}$;
- The partial extraction of all attributes in each attribute instantiation of ι_A^{Ii}, noted $\theta_{pA_{\iota_A}}$, as defined by $\theta_{pA_{\iota_A}} = \bigcup_{i=1}^{n} \theta_{pA_{\iota_{A_j}}}$ and $j \in \mathbb{N}$;
- The partial extraction of all datavalues in the range of each attribute instantiation of ι_A^{Ii}, noted $\theta_{pV_{\iota_A}}$, as defined by $\theta_{pV_{\iota_A}} = \bigcup_{i=1}^{n} \theta_{pV_{\iota_{A_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all instance enumerations of $\sqcup_I$ where I_i is defined as an instance, noted $\sqcup_I^{Ii}$, as defined by $\sqcup_I^{Ii}$={(I_i,... I_n)...(I_i,... I_m)} *with* $I_j \in sI$ *and* $j \in \mathbb{N}$;
- The partial extraction of all the instances different from I_i, enumerated in $\sqcup_I^{Ii}$, noted $\theta_{pI\sqcup I}$, as defined by $\theta_{pI_{\sqcup I}} = \bigcup_{i=1}^{n} \theta_{pI_{\sqcup I\,j}}$ and $j \in \mathbb{N}$;
- The extraction of all value restrictions of $\rho_R(R)$ where I_i is defined as a range, noted ρ_R^{Ii} as defined by ρ_R^{Ii}={(R_1,(I_i)...(R_n, (I_i)} with $R_j \in sR$ and $j \in \mathbb{N}$;
- The partial extraction of all roles defined as range in ρ_R^{Ii}, noted $\theta_{pR\rho_R}$, is defined by $\theta_{pR\rho_R} = \bigcup_{i=1}^{n} \theta_{pR_{\rho_{R_j}}}$ and $j \in \mathbb{N}$.

Definition 31: The **complete datavalue extraction rule** is defined as follows. Given a data value $V_i \in sV$, the complete extraction of V, noted θ_{cVi}={θ_{pVi},$\sqcup_V^{Vi}$, $\theta_{V\sqcup V}$, ι_T^{Vi}, $\theta_{pT\iota_T}$, ρ_A^{Vi}, $\theta_{pA_{\rho_A}}$}, in the extraction result, a set containing :

- The partial extraction of V_i, noted θ_{pVi}, as defined by θ_{piV}={V_i^{θ}};
- The extraction of all the datavalue unions of $\sqcup_V$ where V_i is defined as a datavalue, noted $\sqcup_V^{Vi}$, as defined by $\sqcup_V^{Vi}$={(V, {V1...Vn})} with $V_j \in sV$ and $j \in \mathbb{N}$;
- The partial extraction of all datavalues enumerated with V_i in each datavalue enumeration of $\sqcup_V^{Vi}$, noted $\theta_{V\sqcup V}$, as defined by $\theta_{V\sqcup V} = \bigcup_{i=1}^{n} \theta_{pV_{\sqcup V_j}}$ and $j \in \mathbb{N}$;
- The extraction of all datatype instantiations of ι_T where V_i is defined as a datavalue, noted ι_T^{Ti}, as defined by $\iota_T^{Ti} = \{(T_1, V) \dots (T_n, V)\}$ with $T_i \in sT$ and $j \in \mathbb{N}$;
- The partial extraction of all datatypes of each datatype instantiation of ι_T^{Ti}, noted $\theta_{pT\iota T}$, as defined by $\theta_{pT\iota T} = \bigcup_{i=1}^{n} \theta_{pT_{\iota_{T_j}}}$ and $j \in \mathbb{N}$;
- The extraction of all value restrictions of $\rho_A$where V_i is defined as an datavalue, noted ρ_A^{Vi}, as defined by ρ_A^{Vi}=(A_1, (V_i))...(A_n, (V_i))} with $A_j \in sA$ and $j \in \mathbb{N}$;
- The partial extraction of all attributes defined as attribute in ρ_A^{Vi}, noted $\theta_{pA_{\rho_A}}$, as defined by $\theta_{pA_{\rho_A}} = \bigcup_{i=1}^{n} \theta_{pA_{\rho_{A_j}}}$ and $j \in \mathbb{N}$;

2.4 Partial extraction rules

This part formalizes the **six partial extraction rules** used by the set of complete ones, i.e. **concept, datatype, role, attribute, instance** and **datavalue** partial extractions.

Definition 32: The **partial concept extraction rule** is defined as follows. Given a concept $D_i \in$ sC, a partial extraction of D_i, noted θ_{pDi} ={D_i^{θ}, $\leq_{Ditop}^{\theta}$} in the extraction result, is the set containing the concept D_i itself as defined by D_i^{θ}={*TopConcept, D_i*} and the partial order between the top concept and D_i defined *by* $\leq_{\mathrm{Ditop}}^{\theta}$={(*TopConcept, D_i*)}.

Definition 33: The **partial datatype extraction rule** is defined as follows. Given a data type $T_i \in sT$, the partial extraction of T_i, noted $\theta_{piT}=\{T_i^{\theta}, \leq_{Titop}^{\theta}\}$, in the extraction result, is a set containing T_i itself as defined by $T_i^{\theta}=\{T_i\}$ and the partial order between the top datatype and T_i defined by $\leq_{Titop}^{\theta}=\{(TopDatatype, T_i)\}$.

Definition 34: The **partial role extraction rule** is defined as follows. Given a role $R_i \in sR$, a partial extraction of R_i, noted $\theta_{pRi}=\{ R_i^{\theta}, \leq_{Ritop}^{\theta}, \kappa_R^{Ri}, \mathrm{sK}_R^K, maxCard_{Ri}, minCard_{Ri}\}$ in the extraction result is a set containing:

- The role R_i itself as defined by $R_i^{\theta}=\{TopRole, R_i\}$;
- The partial order between the top role and R_i, noted $\leq_{Ritop}^{\theta}$, as defined by $\leq_{Ritop}^{\theta}=\{(TopRelation, R_i)\}$;
- The extraction of all the role characterisations of κ_R, where R_i is defined as role, noted κ_R^{Ri}, as defined by $\kappa_R^{Ri}=\{(R_i, K_1)...(R_i,K_n)\}$ with $K_j \in sK_R$ and $j \in \mathbb{N}$;
- The extraction of all the role characteristics of κ_R^{Ri} where R_i is defined as role, noted sK_R^K, as defined by $\mathrm{sK}_R^K=\{K_1...K_n\}$ with $K_j \in sK_R$ and $j \in \mathbb{N}$;
- The maximal cardinality of R_i noted $maxCard_{Ri}=(R_i, n)$ with $n \in \mathbb{N}$;
- The minimal cardinality of R_i noted $minCard_{Ri}=(R_i, n)$ with $n \in \mathbb{N}$.

Definition 35: The **partial attribute extraction rule** is defined as follows. Given an attribute $A_i \in sA$, a partial extraction of A_i, noted $\theta_{pAi}=\{ A_i^{\theta}, \leq_{Atop}^{\theta}, \kappa_A^{Ai}, sK_A^K\}$, in the extraction result, is a set containing:

- The extraction of A_i itself as defined by $A_i^{\theta}=\{TopAttribute, A_i\}$;
- The partial order between the top attribute and A_i, noted $\leq_{Aitop}^{\theta}$, as defined by $\leq_{Aitop}^{\theta}=\{(TopAttribute, A_i)\}$;
- The extraction of all the attribute characterizations of κ_A, where A_i is defined as an attribute, noted κ_A^{Ai}, as defined by $\kappa_A^{Ai}=\{(A_i, K_1)\}...(A,K_n)\}$ with $K_j \in sK_A$ and $j \in \mathbb{N}$;
- The extraction of all the attribute characteristics of κ_A^{Ai}, where A_i is defined as an attribute, noted sK_A^K, as defined by $sK_A^K=\{K_1...K_n\}$ with $K_j \in sK_A$ and $j \in \mathbb{N}$;

Definition 36: The **partial instance extraction rule** is defined as follows. Given an instance $I_i \in sI$, the partial extraction of I_i, noted $\theta_{pIi}=\{I_i^{\theta}\}$, in the extraction result, a set containing I_i itself as defined by $I_i^{\theta}=\{I_i\}$.

Definition 37: The **partial datavalue extraction rule** is defined as follows. Given a datavalue $V_i \in sV$, the partial extraction of V_i, noted $\theta_{pVi}=\{V_i^{\theta}\}$, in the extraction result, a set containing V_i itself is defined by $V_i^{\theta}=\{V_i\}$.

2.5 Sub-ontology Global Log of Changes Generation Rule

This part formalizes the two rules respectively used to generate the sub-ontology global log of changes extracted by a view on an ontology and to modify the ontology global log of changes with the addition of the change corresponding to the extraction. The generation of the sub-ontology global log of changes follows the following rule:

Definition 38: The **sub-ontology global log of changes generation rule** is defined as follows:

- Given an ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$, and its global log of changes noted log_o, as defined by $log_o=(log_1, ..., log_n)$ with $log_i=(\beta_1, ..., \beta_n)$ and β_i a basic or complex change, $\beta_i=(\beta_{B1, ..., \beta_{Bn}})$, with $\beta_{Bi} \rightarrow \mathcal{S}_O + a_i$ or $\beta_{Bi} \rightarrow \mathcal{S}_O - a_i$, with a_i an arbitrary axiom of $\mathcal{S}_O$;

- Given the view result $V_{\mathcal{S}_O} = \{\mathcal{S}_O{}^C, \mathcal{S}_O{}^T, \mathcal{S}_O{}^R, \mathcal{S}_O{}^A, \mathcal{S}_O{}^I, \mathcal{S}_O{}^V\}$ of a view V on $\mathcal{S}_O$, with $\mathcal{S}_O{}^{\varepsilon} = \cup_{i=1}^{n} \mathcal{S}_O{}^{\varepsilon i}$ for each starter element $\varepsilon_i \in \{sC, sT, sR, sA, sI, sV\}$ and each starter extraction $\mathcal{S}_O{}^{\varepsilon i} \rightarrow \theta_{c\varepsilon i}$ such that the complete extraction defined by $\theta_{c\varepsilon i} = \cup_{i=1}^{n} a_j^{\varepsilon i}$ corresponds to a set of axioms $a_j^{\varepsilon i}$ where ε_I is used as parameter;

The global log of changes of the sub-ontology $\mathcal{S}_O$' defined by V, noted log_V, is defined by $log_V = (\beta_{V1}, \dots, \beta_{Vn})$ for each change β_{Vi} satisfying $\beta_{Vi} \in log_o$ and $\beta_{Vi} \rightarrow \mathcal{S}_O + a_i$ such that $a_i := a_j^{\varepsilon i}$.

To summarize, a sub-ontology global log of change log_V is a subset of the ontology global log of changes log_o in which addition changes β_{Vi} adds axioms a_i that are equivalent of axioms $a_j^{\varepsilon i}$ in the sub-ontology. Also, the extraction of a view sub-ontology implies the addition of a complex change to the ontology global log of changes, noted *extractSubOntology()*, as defined in the following rule:

Definition 38: The ontology global log of changes view representation rule is defined as follows:

- Given an ontology $\mathcal{S}_O = (\Omega_O, \Sigma_O, \Phi_O, E_O)$, and its global log of changes noted log_o, as defined by $log_o = (log_1, \dots, log_n)$;
- Given the global log of changes of the sub-ontology $\mathcal{S}_O$' defined by a view V, noted $log_V = (\beta_{V1}, \dots, \beta_{Vn})$;

The new global log of change of $\mathcal{S}_O$, noted log_{Onew}, according to the extraction of the view V, is defined by $log_{onew} = (log_O, \beta_{C1})$ with β_{C1} the composed change defined by $\beta_{C1} = extractSubOntology(log_V)$.

In other words, the representation of the extraction of a sub-ontology $\mathcal{S}_O$' defined by a view V from an ontology $\mathcal{S}_O$ is handled by the adding the composed change $extractSubOntology(log_V)$, having the global log of changes of the view log_V as parameter, to the ontology global log of change log_O.

2.6 View Extraction Example

This part illustrates the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process by a view extraction example. The view extraction illustrated in this part is realized on the example ontology $\mathcal{S}_O$, inspired by the UOBM Ontology Benchmark, modelled in Chapter 5 (c.f. G-MOT model (Paquette & Magnan, 2008) on Figure 19 below).

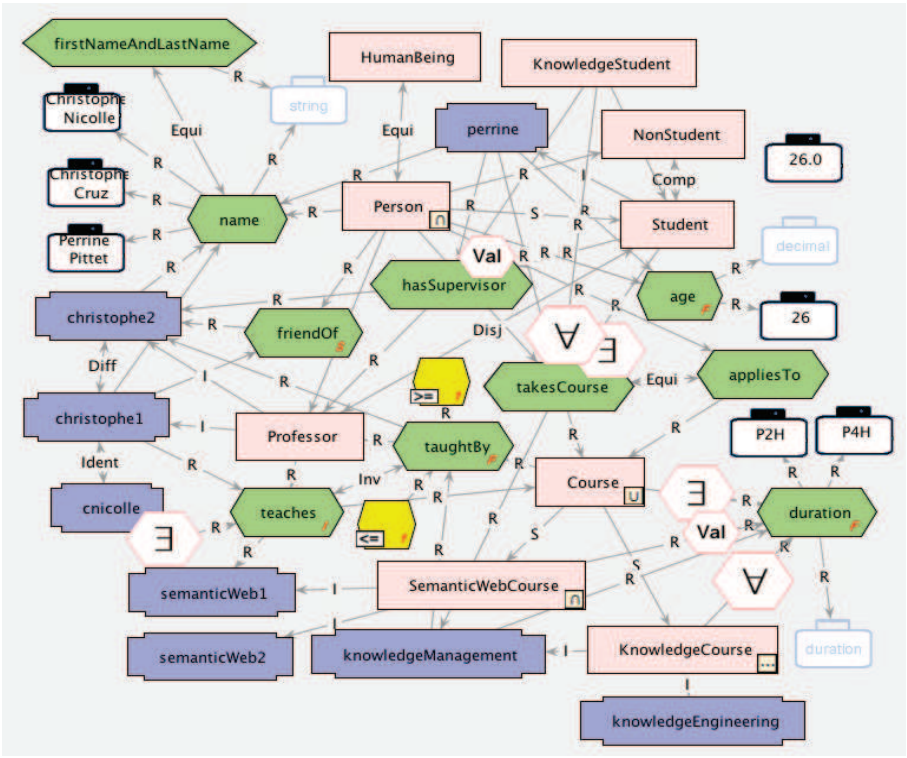


Figure. 19. Graphical Representation of the Ontology Example $\mathcal{S}_O$ with the G-MOT Editor .

2.7 View Extraction Result

A sub-ontology $\mathcal{S}_O'$ is extracted by a view V on $\mathcal{S}_O$, defined by $V(\mathcal{S}_O)=\bigcup_{i=1}^{n} E(\varepsilon_i)_i$ with three starter extraction rules $E(\varepsilon_i)_i$ such as *E(Person)*, *E(teaches)* and *E(christophe1). The* corresponding view result $V_{\mathcal{S}_O}=\{\mathcal{S}_O{}^{Di}, \mathcal{S}_O{}^{Ri}, \mathcal{S}_O{}^{Ii}\}$ obtained with the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process, is defined by the three starter extraction rule results $\mathcal{S}_O{}^{Di}$, $\mathcal{S}_O{}^{Ri}$ and $\mathcal{S}_O{}^{Ii}$, with the respective starter elements: the concept D_i referring to *Person*, the role R_i referring to *teaches* and the instance I_i referring to *christophe1*. $\mathcal{S}_O{}^{Di}$, $\mathcal{S}_O{}^{Ri}$ and $\mathcal{S}_O{}^{Ii}$ are developed according to the following complete extractions: $\mathcal{S}_O{}^{Di}=\{\theta_{cDi}\}$, $\mathcal{S}_O{}^{Ri}=\{\theta_{cRi}\}$ and $\mathcal{S}_O{}^{Ii}=\{\theta_{cIi}\}$ *with:*

- $\theta_{cDi}=\{\theta_{pDi}, (\theta_{pDsup}, \leq^{\theta}_{Disup}), (\theta_{pDsub}, \leq^{\theta}_{Cisup}), (\varepsilon_C^{Di}, \theta_{pD\varepsilon_C}), (-_C^{Di}, \theta_{pD-_C}), (\delta_C^{Di}, \theta_{pD\delta_C}), (\sqcap_C^{Di}, \theta_{pD\sqcap_C}), (\sqcup_C^{Di}, \theta_{pD\sqcup_C}), \sigma_R^{Di}, \theta_{pR_{\sigma_R}}, \theta_{pD_{\sigma_R}}, \sigma_A^{Di}, \theta_{pD_{\sigma_A}}, \theta_{pT_{\sigma_A}}, \rho_{\forall R}^{Di}, \theta_{pR\rho_{\forall R}}, \rho_{\exists R}^{Di}, \theta_{pR\rho_{\exists R}}\}$:
  - $\theta_{pDi}=\{D_i^{\theta}, \leq^{\theta}_{Ditop}\}=\{\{TopConcept, Person\}, \{(TopConcept, Person)\}\}$;
  - $\leq^{\theta}_{Dsiup}=\{TopConcept, Person\}$;
  - $\theta_{pDsup}=\bigcup_{i=1}^{n}\theta_{pD_{Sup_i}}=\{D_{sup}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{TopConcept\}, \{(TopConcept, TopConcept)\}\}$;
  - $\leq^{\theta}_{Disub}=\{(Person, Professor), (Person, Student)\}$;
  - $\theta_{pDsub}=\bigcup_{i=1}^{n}\theta_{pD_{Sub_i}}$ *with :*
    - $\theta_{pD_{sub_{C1}}}=\{D_{sub}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{TopConcept, Professor\}, \{(TopConcept, Professor)\}\}$;
    - $\theta_{pD_{sub_{C2}}}=\{D_{sub}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{TopConcept, Student\}, \{(TopConcept, Student)\}\}$;
  - $\varepsilon_C^{Di}=\{(Person, HumanBeing)\}$;
  - $\theta_{pD\varepsilon_C}=\bigcup_{i=1}^{n}\theta_{pD_{\varepsilon_{Ci}}}=\theta_{pD_{\varepsilon_{C1}}}=\{D_{\varepsilon C1}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{\{\{TopConcept, HumanBeing\}, \{(TopConcept, HumanBeing)\}\}\}$;
  - $-_C^{Di}=\{\}$;
  - $\theta_{pD-_C}=\{\}$;
  - $\delta_C^{Di}=\{(Person, Course)\}$;
  - $\theta_{pD\delta_C}=\bigcup_{i=1}^{n}\theta_{pD_{\delta_{Ci}}}=\theta_{pD_{\delta_{C1}}}=\{D_{\delta C1}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{\{TopConcept, Course\}, \{(TopConcept, Course)\}\}$;
  - $\sqcap_C^{Di}=\{\}$;
  - $\theta_{pD\sqcap_C}=\{\}$;
  - $\sqcup_C^{Di}=\{\}$;
  - $\theta_{pD\sqcup_C}=\{\}$;
  - $\sigma_R^{Di}=\{(friendOf, (Person, Person))\}$;
  - $\theta_{pR_{\sigma_R}}=\bigcup_{i=1}^{n}\theta_{pR_{\sigma_{R_i}}}$ *and* $\theta_{pR_{\sigma R1}}=\{R_{\sigma R}^{\theta}, \leq^{\theta}_{Rtop}, \kappa_R^R, sK_R^K, maxCard_R^n, minCard_R^n\}$ *with:*
    - $R_{\sigma}^{\theta}=\{TopRole, friendOf\}$;
    - $\leq^{\theta}_{Rtop}=\{(TopConcept, friendOf)\}\}$;
    - $\kappa_R^R=\{(friendOf, Symmetric)\}\}$;
    - $sK_R^K=\{Symmetric\}$;
    - $maxCard_R^n=\{\}$;
    - $minCard_R^n=\{\}$;
  - $\theta_{pD_{\sigma_R}}=\bigcup_{i=1}^{n}\theta_{pD_{\sigma_{R_i}}}=\theta_{pD_{\sigma_{R_1}}}=\{D_{\sigma R1}^{\theta}, \leq^{\theta}_{Dtop}\}=\{\{\{\{TopConcept, Person\}, \{(TopConcept, Person)\}\}\}$;
  - $\sigma_A^{Di}=\{(name, (Person, xsd:string)), (age, (Person, xsd:decimal))\}$;
  - $\theta_{pA_{\sigma_A}}=\bigcup_{i=1}^{n}\theta_{pA_{\sigma_{A_i}}}$ *with* $\theta_{pA_{\sigma_{A1}}}$ *and* $\theta_{pA_{\sigma_{A2}}}$ *such that:*
    - $\theta_{pA_{\sigma_{A1}}}=\{A_{\sigma A1}^{\theta}, \leq^{\theta}_{Atop}, \kappa_A^A, sK_A^K\}$ *with:*
      - $A_{\sigma A1}^{\theta}=\{TopAttribute, name\}$;
      - $\leq^{\theta}_{Atop}=\{(TopAttribute, name)\}$;
      - $\kappa_A^A=\{\}$;
      - $sK_A^K=\{\}$;
    - $\theta_{pA_{\sigma_{A2}}}=\{A_{\sigma A2}^{\theta}, \leq^{\theta}_{Atop}, \kappa_A^A, sK_A^K\}$ *with:*
      - $A_{\sigma A2}^{\theta}=\{TopAttribute, age\}$;
      - $\leq^{\theta}_{Atop}=\{(TopAttribute, age)\}$;

- κ_A^A={};
- sK_A^K={};

- $\theta_{pT_{\sigma_A}} = \bigcup_{i=1}^{n} \theta_{pT_{\sigma_{A_i}}}$ *with* $\theta_{pT_{\sigma_{A_1}}}$ *and* $\theta_{pT_{\sigma_{A_2}}}$ *such that*:
  - $\theta_{pT_{\sigma_{A_1}}}$={ $T^{\theta}_{\sigma A1}$,$\leq^{\theta}_{Ttop}$ }={{TopType, xsd:string}, {(TopType, xsd:string)}};
  - $\theta_{pT_{\sigma_{A_2}}}$={ $T^{\theta}_{\sigma A2}$,$\leq^{\theta}_{Ttop}$ }= {{TopType, xsd:decimal},{(TopType, xsd:decimal)}};
- $\rho^{Di}_{\forall R}$={};
- $\theta_{pR\rho_{\forall R}}$={};
- $\rho^{Di}_{\exists R}$={};
- $\theta_{pR\rho_{\exists R}}$={}.

Figure 20 shows a graphical representation of ${\mathcal{S}_O}^{Di}$ realized with the G-MOT Ontology Editor.

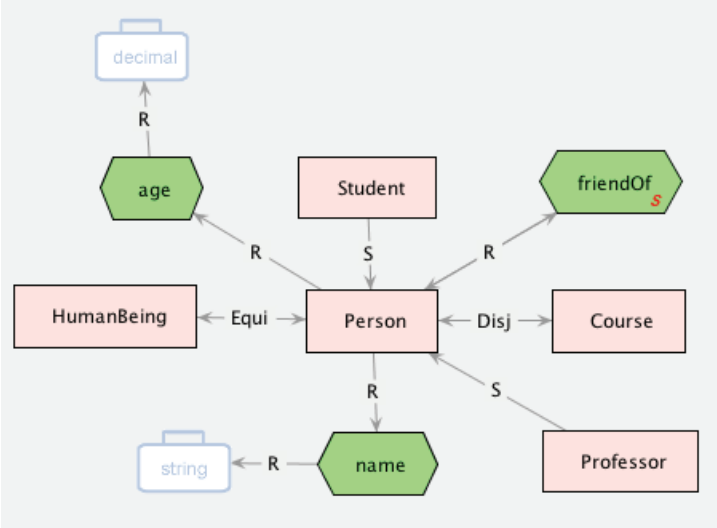


Figure 20. Starter Concept *Person* Extraction Result

θ_{cRi}={ θ_{pRi}, ε_R^{Ri}, $\theta_{pR\varepsilon_R}$, $\leq^{\theta}_{Risup}$, $\theta_{pRsup,}$ $\leq^{\theta}_{Risub}$, θ_{pRsub}, $-_R^{Ri}$, θ_{pR-R}, σ_R^{Ri}, $\theta_{pD\sigma R}$, $\rho^{Ri}_{\forall R}$, $\theta_{pD\rho_{\forall R}}$, $\rho^{Ri}_{\exists R}$, $\theta_{pD\rho_{\exists R}}$, ρ_R^{Ri}, $\theta_{pI\rho R}$} *with:*

- θ_{pRi}={ R_i^{θ},$\leq^{\theta}_{Ritop}$, κ_R^{Ri}, sK_R^K, $maxCard_{Ri}$, $minCard_{Ri}$} *with:*
  - R_i^{θ} ={TopRole,teaches};
  - $\leq^{\theta}_{Ritop}$=(TopConcept,teaches)}};
  - κ_R^{Ri}={(teaches, InverseFunctional)};
  - sK_R^K={InverseFunctional};
  - $maxCard_{Ri}$={};
  - $minCard_{Ri}$={};
- ε_R^{Ri}={};
- $\theta_{pR\varepsilon}$={};
- $\leq^{\theta}_{Risup}$={};
- θ_{pRsup}={};
- $\leq^{\theta}_{Risub}$={};
- θ_{pRsub}={};
- $-_R^{Ri}$={(teaches, isTaughtBy)};
- θ_{pR-R}=$\bigcup_{i=1}^{n} \theta_{pR-Ri}$=$\theta_{pR-R1}$ *and* θ_{pR-R1}={ R^{θ}_{-R1},$\leq^{\theta}_{Rtop}$, κ_R^R, sK_R^K, $maxCard_R^n$, $minCard_R^n$ } *with:*
  - R^{θ}_{-R1} ={{TopRole,isTaughtBy};
  - $\leq^{\theta}_{Rtop}$={(TopConcept,isTaughtBy)};
  - κ_R^R={(isTaughtBy, Functional)};
  - sK_R^K={Functional};
  - $maxCard_R$={(isTaughtBy,1)};
  - $minCard_R$={(isTaughtBy, 1)};

- σ_R^{Ri}={(teaches, (Professor, Course))} ;
- $\theta_{pC\sigma_R} = \bigcup_{i=1}^{n} \theta_{pC_{\sigma R_i}}$ with $\theta_{pC_{\sigma R_1}}$ and $\theta_{pC_{\sigma R_2}}$ such that:
  - $\theta_{pC_{\sigma R_1}} = \{ D^{\theta}_{\sigma R1}, \leq^{\theta}_{Dtop} \} = \{\{\{TopConcept, Professor\}, \{(TopConcept, Professor)\}\}$;
  - $\theta_{pC_{\sigma R_2}} = \{D^{\theta}_{\sigma R2}, \leq^{\theta}_{Dtop} \} = \{\{TopConcept, Course\}, \{(TopConcept, Course)\}\}$;
- $\rho_{\forall R}^{Ri}$={};
- $\theta_{pD\rho_{\forall R}}$={};
- $\rho_{\exists R}^{Ri}$={(teaches,Course)};
- $\theta_{pD\rho_{\exists R}} = \bigcup_{i=1}^{n} \theta_{pC_{\rho \exists R_i}}$ with $\theta_{pC_{\rho \exists R_1}}$ such that:
  - $\theta_{pC_{\rho \exists R_1}} = \{ D^{\theta}_{\rho \exists R_1}, \leq^{\theta}_{Dtop} \} = \{\{TopConcept, Course\}, \{(TopConcept, Course)\}\}$;
- ρ_R^{Ri}={};
- $\theta_{pI\rho_R}$={};

Figure 21 shows a graphical representation of $\mathcal{S}_O^{Ri}$ realized with the G-MOT Ontology Editor.

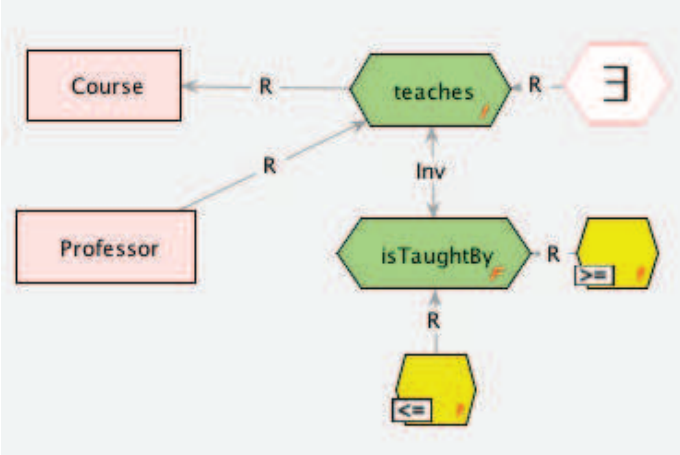


Figure 21. Starter Role *teaches* Extraction Result

θ_{cli}= {θ_{pIi}, ι_C^{Ii}, $\theta_{pD\iota_C}$, ι_R^{Ii}, $\theta_{pR_{\iota R}}$, $\theta_{pI_{\iota R}}$, ι_A^{Ii}, $\theta_{pA_{\iota A}}$, $\theta_{pV_{\iota A}}$, ε_I^{Ii}, $\theta_{pI_{\varepsilon_I}}$, δ_I^{Ii}, $\theta_{pI_{\delta_I}}$, $\sqcup_I^{Ii}$, $\theta_{pI_{\sqcup I}}$, ρ_R^{Ii}, $\theta_{pR\rho R}$} with:

- θ_{pIi}=I_i^{θ}={christophe1};
- ι_C^I={(Professor, christophe1)},
- $\theta_{pD\iota C} = \bigcup_{i=1}^{n} \theta_{pD_{\iota C_i}} = \theta_{pD_{\iota C_1}} = \{ D^{\theta}_{\iota C1}, \leq^{\theta}_{Dtop} \} = \{\{TopConcept, Professor\}, \{(TopConcept, Professor)\}\}$;
- ι_R^{Ii}={(friendOf,(christophe1, christophe2)), (teaches, (christophe1, semanticWeb1))};
- $\theta_{pR_{\iota R}} = \bigcup_{i=1}^{n} \theta_{pR_{\iota R_i}}$
  - with $\theta_{pR_{\iota R1}} = \{ R^{\theta}_{\iota R1}, \leq^{\theta}_{Rtop}, \kappa_R^R, sK_R^K, maxCard_R^n, minCard_R^n \}$ such that:
    - $R^{\theta}_{\iota R1}$ ={{TopRole,friendOf};
    - $\leq^{\theta}_{Rtop}$={(TopConcept,friendOf)}};
    - κ_R^R={(friendOf, Symmetric)}};
    - sK_R^K={Symmetric};
    - $maxCard_R$={};
    - $minCard_R$={};
  - and $\theta_{pR_{\iota R2}} = \{ R^{\theta}_{\iota R2}, \leq^{\theta}_{Rtop}, \kappa_R^R, sK_R^K, maxCard_R^n, minCard_R^n \}$ such that:
    - $R^{\theta}_{\iota R2}$ ={{TopRole,teaches};
    - $\leq^{\theta}_{Rtop}$={(TopConcept,teaches)}};
    - κ_R^R={(teaches, InverseFunctional)}};
    - sK_R^K={InverseFunctional};
    - $maxCard_R$={};

- $minCard_R$={};
- $\theta_{pI_{\iota_R}}$={christophe2, semanticWeb1};
- ι_A^{Ii}={(name,(christophe1, "Christophe Nicolle"))};
- $\theta_{pA_{\iota_A}}=\cup_{i=1}^{n}\theta_{pA_{\iota_{Ai}}}=\theta_{pA_{\iota_{A1}}}$ with $\theta_{pA_{\iota_{A1}}}=\{A_{\iota_{A1}}^{\theta}, \leq_{Atop}^{\theta}, \kappa_A^A, sK_A^K\}$ such that:
  - $A_{\iota_{A1}}^{\theta}$={TopAttribute, name};
  - $\leq_{Atop}^{\theta}$={(TopAttribute, name)};
  - κ_A^A={};
  - sK_A^K={};
- $\theta_{pV_{\iota_A}}$={"Christophe Nicolle"};
- ε_I^{Ii} = {(christophe1, cnicolle)};
- $\theta_{pI_{\varepsilon_I}}=\cup_{i=1}^{n}\theta_{pI_{\varepsilon_{I_i}}}=\theta_{pI_{\varepsilon_{I1}}}$ with $\theta_{pI_{\varepsilon_{I1}}}$={$sI^{\theta}$}={cnicolle};
- δ_I^{Ii}= {(christophe1, christophe2)};
- $\theta_{pI_{\delta_I}}=\cup_{i=1}^{n}\theta_{pI_{\delta_{I_i}}}=\theta_{pI_{\delta_{I1}}}$ with $\theta_{pI_{\delta_{I1}}}$={$sI^{\theta}$}={christophe2};
- $\sqcup_I^{Ii}$={};
- $\theta_{pI_{\sqcup I}}$={};
- ρ_R^{Ri}={};
- $\theta_{pI_{\rho_R}}$={};

Figure 22 shows a graphical representation of $\mathcal{S}_O{}^{Ii}$ realized with the G-MOT Ontology Editor.

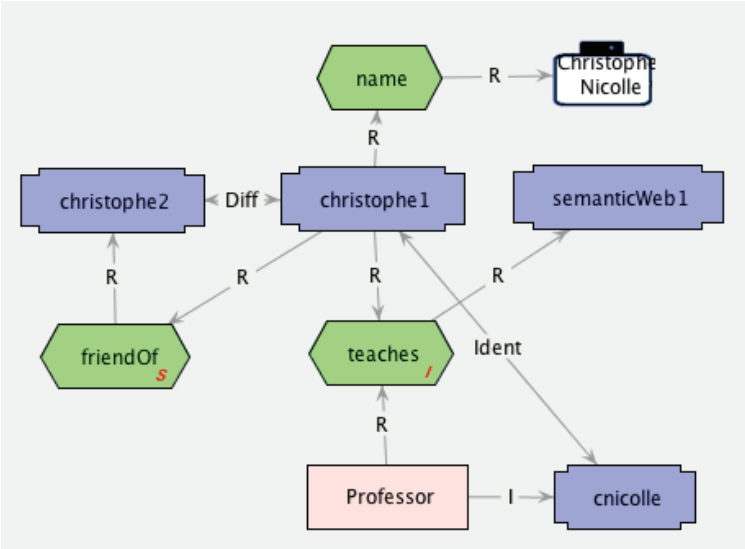


Figure 22. Starter Instance *christophe1* Extraction Result

The sub-ontology $\mathcal{S}_O$'=(Ω_O', Σ_O', Φ_O', E_O) corresponding to the view result $V_{\mathcal{S}_O}$={$\mathcal{S}_O{}^{Di}$, $\mathcal{S}_O{}^{Ri}$, $\mathcal{S}_O{}^{Ii}$} is described below:

- Ω_O'={($sC, \leq_C$),($sT, \leq_T$),($sR, \leq_R$),($sA, \leq_A$), sI, sV, sK_R, sK_A,} with:
  - sC =TopConcept, Person, Professor, Student, HumanBeing, Course;
  - $\leq_C$={(TopConcept, Person), (Person, Professor), (Person, Student), (TopConcept, HumanBeing),(TopConcept, Course)};
  - sT={xsd:string, xsd:decimal};
  - $\leq_T$={(TopType, xsd:string), (TopType, xsd:decimal)};
  - sR={TopRole, friendOf, teaches, isTaughtBy};
  - $\leq_R$={(TopRole, friendOf), (TopRole, teaches),(TopRole, isTaughtBy)};
  - sA={TopAttribute, name, age};
  - $\leq_A$ ={(TopAttribute, name), (TopAttribute, age)};
  - sI={christophe1, christophe2, cnicolle, semanticWeb1};

- sV={"Christophe Nicolle"};
- sK_R={Functional, InverseFunctional, Symmetric};
- sK_A={};

- Σ_O'={σ_R , σ_A} with:

  - σ_R={(teaches, (Professor, Course)), (friendOf, (Person, Person))} ;
  - σ_A={(age, (Person, xsd:decimal)), (name, (Person, xsd:string))} ;

- Φ_O'={ι_C, ι_T, ι_R, ι_A, κ_R, κ_A, ε_C, ε_R, ε_A, ε_I, δ_C, δ_I, $-_C$, $-_R$, $maxCard_R$, $minCard_R$, $\sqcap_C$, $\sqcup_C$, $\sqcup_I$, $\sqcup_V$, $\rho_{\exists R}$, $\rho_{\forall R}$, ρ_R, $\rho_{\exists A}$, $\rho_{\forall A}$, ρ_A} with:

  - ι_C={(Professor, christophe1)};
  - ι_T={};
  - ι_R={(friendOf,(christophe1, christophe2)), (teaches, (christophe1, semanticWeb1))};
  - ι_A={(name,(christophe1, "Christophe Nicolle"))};
  - κ_R={(teaches, InverseFunctional), (isTaughtBy, Functional), (friendOf, Symmetric)}};
  - κ_A={};
  - ε_C={(Person, HumanBeing)};
  - ε_R={};
  - ε_A={};
  - ε_I={};
  - δ_C={(Person, Course)};
  - δ_I={(christophe1, christophe2)};
  - $-_C$={};
  - $-_R$={(teaches, isTaughtBy)};
  - $maxCard_R$={(isTaughtBy,1)};
  - $minCard_R$={(isTaughtBy,1)};
  - $\sqcap_C$={};
  - $\sqcup_C$={};
  - $\sqcup_I$={};
  - $\sqcup_V$={};
  - $\rho_{\exists R}$={(teaches, (Course));
  - $\rho_{\forall R}$={};
  - ρ_R={};
  - $\rho_{\exists A}$={};
  - $\rho_{\forall A}$={};
  - ρ_A={};

E_O = {TopConcept, BottomConcept, TopAttribute, BottomAttribute, TopRole, BottomRole, TopDataType, BottomDataType}

Figure 23 shows a graphical representation of $\mathcal{S}_O$' realized with the G-MOT Ontology Editor.

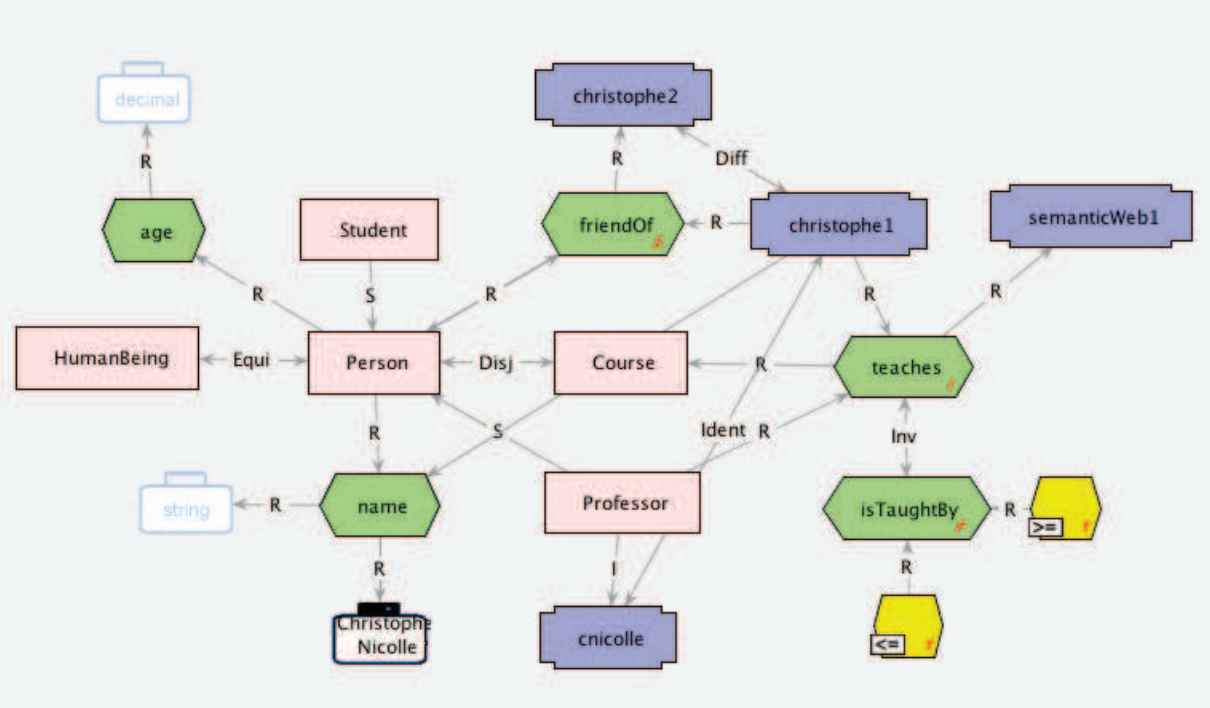


Figure 23: Sub-Ontology S_0' Extraction Result

2.8 View and Ontology Global Logs of Changes

Due to lack of space, the ontology S_0 global log of changes log_0 , which is used to generate the view global log of changes log_V, is not represented here. However, following the generation rule of the sub-ontology log of changes, the extraction of the view may generate the following global log of changes:

log_V=(
(addConcept, (Person)),
(addConcept, (Professor)),
(addConcept, (Student)),
(addConcept, (HumanBeing)),
(addConcept, (Course)),
(addSubConcept, (Professor, Person)),
(addSubConcept, (Student, Person)),
(addDatatype, (xsd:string)),
(addDatatype, (xsd:decimal)),
(addRole, (friendOf)),
(addRole(teaches)),
(addRole, (isTaughtBy)),
(addAttribute, (name)),
(addAttribute, (age)),
(addInstance, (christophe1)),
(addInstance, (christophe2)),
(addInstance, (cnicolle)),
(addInstance, (semanticWeb1)),
(addDataValue, ("Christophe Nicolle")),
(addRoleSignature, (teaches, (Professor, Course))),
(addRoleSignature, (friendOf, (Person, Person))),

(addAttributeSignature, (name, (Person, xsd:string))),
(addAttributeSignature, (age, (Person, xsd:decimal))),
(addConceptInstantiation, (christophe1, Professor)),
(addRoleInstantiation, (christophe1, christophe2, friendOf)),
(addRoleInstantiation, (christophe1, semanticWeb1, teaches)),
(addAttributeInstantiation, (christophe1, "Christophe Nicolle", name)),
(addRoleInverseFunctionalProperty, (teaches)),
(addRoleFunctionalProperty, (isTaughtBy)),
(addRoleSymmetricProperty, (friendOf)),
(addEquivalentConcept, (Person, HumanBeing)),
(addDisjointConcept, (Person, Course)),
(addDifferentInstance, (christophe1, christophe2)),
(addInverseRole, (teaches, isTaughtBy)),
(addRoleMaxCardinality, (isTaughtBy, 1)),
(addRoleMinCardinality, (isTaughtBy, 1)),
(addRoleExistentialRestriction, (teaches, Course))
)

Following the ontology global log of changes view representation rule, the ontology global log of changes will be updated as follow: $log_{Onew}=(log_O, (extractSubOntology(log_V)))$.

2.9 Discussion

According to the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process, a view is defined as a **set of extractions** centred around selected elements of a $\mathcal{SHOIN}(\mathcal{D})$ ontology, called **starters elements**. The three levels of extraction of a view are defined in the form of rules . The first level, i.e. the starter extraction rules , taking as input a starter element, which can be either a **concept,** or a **datatype**, or a **role**, or an **attribute**, or an **instance** or a **datavalue**. They allow defining the second level of extraction, i.e. **complete extraction rules**. Each complete extraction it is composed of the axioms dependent to the starter element and of **partial extractions** of elements used by these axioms. These partial extractions are the third level of extraction and include only the strict definition of the element partially extracted. This ensures producing a finite and **minimal set of axioms** around the starter element. As the view extraction process has been completely formalized from the $\mathcal{SHOIN}(\mathcal{D})$ ontology model, two main properties can qualify the approach: **completeness** and **consistency** (Property 2 and 3).

Property 2: Completeness of the $\mathcal{SHOIN}(\mathcal{D})$ view extraction. The $\mathcal{SHOIN}(\mathcal{D})$ view extraction process, extracting a sub-ontology $\mathcal{S}_O$' defined by a view V on a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O$ is complete in the sense that it allows extraction of any axiom of the ontology as defined by the $\mathcal{SHOIN}(\mathcal{D})$ ontology model.

In other words, a sub-ontology extracted according to the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process is a true $\mathcal{SHOIN}(\mathcal{D})$ ontology.

Property 3: Consistency of the $\mathcal{SHOIN}(\mathcal{D})$ view extraction. Considering a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O$ and a sub-ontology $\mathcal{S}_O$' extracted from $\mathcal{S}_O$ according to the $\mathcal{SHOIN}(\mathcal{D})$ view extraction process, if $\mathcal{S}_O$ is structurally and logically consistent then $\mathcal{S}_O$' is also a structurally and logically consistent $\mathcal{SHOIN}(\mathcal{D})$ ontology.

As stated in first section, the logical consistency of the extracted sub-ontology is ensured by the fact that a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology defined by a view with such process is a subset of a $\mathcal{SHOIN}(\mathcal{D})$ ontology consistent set of axioms, and by the monotonicity property of $\mathcal{SHOIN}(\mathcal{D})$. Its structural consistency is guarantied by the definition of extraction rules compliant with the $\mathcal{SHOIN}(\mathcal{D})$ structural constraints.

Finally, rules have been formalized to produce the **global log of changes of an extracted sub-ontology** and to **represent the extraction in the related source ontology log of changes**. $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies can therefore be managed within the OntoVersionGraph methodology.

3 Conclusion

This chapter has presented a **consistent $\mathcal{SHOIN}(\mathcal{D})$ view extraction process** within the OntoVersionGraph methodology. The approach for extracting views from a $\mathcal{SHOIN}(\mathcal{D})$ ontology allows producing **$\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**, describing a more restricted domain than the ontology source. The **view definition** is used to define this subdomain. It is handled by **selecting the source ontology main elements** to keep in the sub-ontology, i.e. **starter elements**. Each starter element **related sets of axioms are automatically extracted** to generate the corresponding subset from the ontology. **$\mathcal{SHOIN}(\mathcal{D})$ structural constraints** are defined to ensure structural consistency of the extracted subsets during the extraction. **Three levels of extractions** are modelled to allow the production of **minimal subsets**, thereby limiting the sub-domain described by the view. The view evolution and versioning can be performed in the same manner as ontologies in OntoVersionGraph. A **global log of changes** is generated for each view from the view result of the view and the ontology global log of changes, in order to display these features. Finally the **extraction of a view is represented in the ontology global log of changes** as a complex change, to allow managing the versioning of the ontology and its related views. The view extraction process has been completely formalized from the $\mathcal{SHOIN}(\mathcal{D})$ ontology model defined in Chapter 4. In addition, as the sub-ontology extracted by a view is a set of $\mathcal{SHOIN}(\mathcal{D})$ axioms, which is a subset of a $\mathcal{SHOIN}(\mathcal{D})$ ontology set of axioms, **completeness,** and **structural** and **logical consistency** of the sub-ontology are ensured. An application example of this approach has illustrated the extraction of a view from the example ontology modelled in Chapter 4.

This chapter has then completed the formalization of the second block of my thesis proposal consisting in modeling the **logical extraction and change management of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. The change propagation phase is the last phase requiring a specific approach in the OntoVersionGraph methodology. Corresponding to the third block of my proposal, it has to assess the impact and spread the change implemented to the ontology views and/or dependent ontologies. The third block is therefore presented in the following chapter.

References

Noy, N. F., & Musen, M. A. (2004). Specifying ontology views by traversal. In The Semantic Web–ISWC 2004 (pp. 713-725). Springer Berlin Heidelberg.

Obrst, L., Liu, H., & Wray, R. (2003). Ontologies for corporate web applications. AI Magazine, 24(3), 49.

Parsia, B., & Sirin, E. (2004, November). Pellet: An owl dl reasoner. In Third International Semantic Web Conference-Poster (Vol. 18).

Seidenberg, J., & Rector, A. (2006, May). Web ontology segmentation: analysis, classification and use. In Proceedings of the 15th international conference on World Wide Web (pp. 13-22). ACM.

Chapter 7

Change Impact Management of $\mathcal{SHOIN}(\mathcal{D})$ Ontology & Views

Summary

This chapter formalizes the block (3) of my thesis proposal, i.e. modelling the **impact management process of a consistent change propagation phase** within the OntoVersionGraph evolution process. The impact management process ensures two types of consistent change propagation: first, the propagation from a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views, called the **Top-Down Propagation approach**, and, second, from a sub-ontology defined by a view on a source ontology to this ontology and to the other related views, called the **Bottom-Up Propagation approach**. The Top-Down Propagation approach and the Bottom-Up Propagation approach are respectively formalized in the first and the second sections. A third section illustrates the use of the two approach types by applying the whole methodology on two examples.

Plan

1 Top-Down Propagation Approach 178
1.1 Ontology Impact Rule 179
1.2 View Impact Rule 179
1.3 View Update Rule 180
1.4 Sub-Ontology Update Rule 180
1.5 Discussion 181
2 Bottom-Up Propagation Approach 182
2.1 SHOIND Ontology Update Rule 183
2.2 Source Ontology Evolution 183
2.3 Source Ontology Changes Propagation 184
2.4 Discussion 184
3 Change Impact Management Example 185
3.1 Assertional Change Example 185
3.1.1 Introduction to the problem 185
3.1.2 Architect Change Management Process: Round 1 187
3.1.3 Electrician Change Management Process 188
3.1.4 Architect Change Management Process: Round 2 190
3.2 Terminological Change 190
3.3 Discussion 193
4 Conclusion 194

Previous chapter has presented the **logical specification and extraction process of views from a $\mathcal{SHOIN}(\mathcal{D})$ ontology, producing $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies** according to specific domain users assumptions. It has completed the second block of my thesis proposal and allows defining the third one, i.e. the **impact management process of a consistent change propagation between a $\mathcal{SHOIN}(\mathcal{D})$ ontology and its views,** within the OntoVersionGraph evolution process. The impact management process is part of the change propagation phase of the ontology evolution process described in OntoVersionGraph (c.f. Chapter 4 Section 1).

When domain experts want to apply changes on an ontology, a complete ontology evolution process is launched. However if this ontology is the source of one or multiple sub-ontologies defined by views, its evolution has an impact on them. To keep them up-to-date with the evolved ontology, **changes have to be propagated and applied on these sub-ontologies**. Inversely when sub-domain experts want to apply changes on a sub-ontology extracted by a view on the ontology, a first ontology evolution process is launched for the sub-ontology. Before validating and propagating these changes to the other views, the **logical consistency of the whole source ontology has to be assessed** in order to ensure these changes do not cause logical inconsistencies in the other sub-ontologies. Changes have therefore to be propagated to the ontology before being applied on the other sub-ontologies. Impact management of these two types of propagation is the role of the **change propagation phase** proposed in the OntoVersionGraph methodology.

This chapter is articulated in three sections. A first section presents the Top-Down approach ensuring **a consistent propagation of changes implemented on a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views**, allowing to consistently update the related sub-ontologies. A second section presents the Bottom-Up approach to **propagate changes from an evolved view to its source ontology and the from the ontology to its other views**. A third section illustrates the entire impact management process on a example of change propagation within a knowledge base using an ontology to describe a building and its different domain experts' views.

1 Top-Down Propagation Approach

This section presents the **Top-Down Propagation approach** to formalize the propagation of changes from a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views. A first part defines the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **ontology impact rule** allowing to produce the changes subset to propagate its views. A second part defines the **view impact rule** assessing the impact of this subset of changes on the view definition. A third part defines the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **view update rule** used to update the view definition if impacted. A fourth part defines the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **sub-ontology update rule** used to extract the sub-ontology according to its view definition, impacted or not, from the evolved ontology.

The Top-Down Propagation approach follows five steps illustrated in Figure 24 using these four rules.

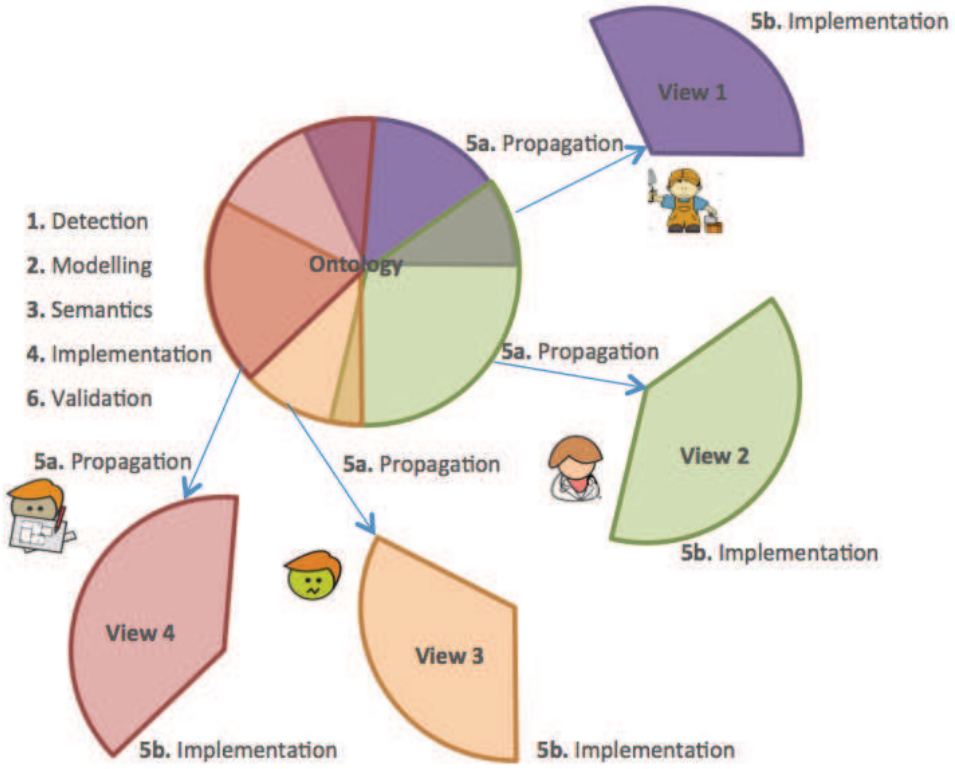


Figure 24. Top-Down Propagation Approach Steps

First an **ontology evolution process** is launched in order to model changes required by the domain experts on a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_0$ (c.f. 1.Detection on Fig.24). The **change modelling phase** allows the domain experts to represent the detected changes in a log of changes, noted log_0, and to correct them in order to ensure their structurally consistent application (c.f. 2.Modelling on Fig. 24). The **change semantics phase** develops the changes modelled in log_0 into basic ones to apply a logical consistency preventive analysis on them (c.f. 3.Semantics on Fig. 24). This analysis returns the **minimal inconsistent set of changes** identified in log_0 in order to guide the domain experts resolve logical inconsistencies within a new change modelling phase. When changes are assessed logically consistent they are implemented on a copy of $\mathcal{S}_0$, noted $\mathcal{S}_{01}$, during the **change implementation phase**, wherein a second logical consistency check is realized on the evolved copy by using the **Pellet reasoner** (c.f. 4.Implementation on Fig. 24). At the **change propagation phase** (c.f. 5a.Propagation on Fig. 24), propagation of these changes can then be handled to **manage their impact on the ontology views** V_i. In

parallel, the ontology continues its evolution process through the validation phase (c.f. 6.Validation on Fig. 24). The evolution of $\mathcal{S}_O$ is noted $\mathcal{S}_O{}^{new}$.

Back to the propagation phase, the **set of elements of the ontology impacted by deletion changes is evaluated** and processed according to the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **ontology impact rule** (c.f. Definition 39). It allows identifying which ontological elements of $\mathcal{S}_O$, potentially contained by the sub-ontologies $\mathcal{S}_O{}'_i$ of each view V_i have been deleted to produce $\mathcal{S}_O{}^{new}$. It constitutes the second step of the Top-Down Propagation approach.

Third, the set of subset extractions, composing each view result $V_{i\mathcal{S}_O}$ of the views V_i defined on $\mathcal{S}_O$ and, which are impacted by these deletion changes, is evaluated and processed according to the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **view impact rule** (c.f. Definition 40).

Fourth, **each view definition** $V_i(\mathcal{S}_O)$ **is updated** according to its corresponding set of impacted extractions as described by the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **view update rule** (c.f. Definition 41).

Fifth, the extraction of the sub-ontologies corresponding to each updated view definition is realized on the evolved copy $\mathcal{S}_O{}^{new}$ of $\mathcal{S}_O$ according to the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **sub-ontology update rule** (c.f. Definition 42). Propagated changes are then implemented on the whole set of sub-ontologies $\mathcal{S}_O{}'_i$ (c.f. 5b.Implementation on Fig.21).

1.1 Ontology Impact Rule

This part defines the **ontology impact rule, identifying the set of ontological elements impacted by deletion changes in a log of changes.** Addition changes are not considered, as they do not structurally impact the sub-ontology definition, which is built upon the existing ontology axioms.

Considering a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O=(\Omega_O, \Sigma_O, \Phi_O, E_O)$, the ontology impact rule determines the set Ω_O^{Imp} of elements $\varepsilon_i \in \{sC, sT, sR, sA, sI, sV\}, \subseteq \Omega_O$ impacted by the set of deletion changes B_{logi}^- identified in a consistent log of changes log_i issue from the change semantic phase of the i^{th} evolution of $\mathcal{S}_O$.

Definition 39. $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **Ontology Impact Rule**.

Given log_i, the log of changes issued from the semantic phase of the i^{th} ontology evolution of the ontology $\mathcal{S}_O$, in which changes are all decomposed as basic changes $\beta_{Bi}=(\pi_j, (\varepsilon_1, \ldots, \varepsilon_n)), \pi \in \Pi$, with $\varepsilon_i \in \{sC, sT, sR, sA, sI, sV\}$ and Π the set of all $\mathcal{SHOIN}(\mathcal{D})$ change operator types and $j \in \mathbb{N}$; the **set of deletion changes, noted** $\boldsymbol{B_{logi}^-}$, is composed of the basic changes $\beta_{Bi} \in log_0$, in which the change operator type π_j is assumed as a deletion operator of an ontological element ε_i.

Given B_{logi}^-, the **set** $\boldsymbol{\Omega_O^{Imp}}$ **of elements of** $\boldsymbol{\mathcal{S}_O}$ **impacted by deletion changes** is composed of ontological elements $\varepsilon_i^- \in \{sC, sT, sR, sA, sI, sV\}$ used as parameters in changes of B_{logi}^-, such that $\Omega_O^{Imp}=\bigcup_{i=1}^{n}(\varepsilon_i^-)$.

In other words, the set Ω_O^{Imp} of ontological elements ε_i^- of $\mathcal{S}_O$ impacted by deletion changes, are those appearing as parameters of the changes, in which the change operator type is one of the following: deleteConcept, deleteDatatype, deleteRole, deleteAttribute, deleteInstance and deleteDatavalue.

1.2 View Impact Rule

This part defines the $\boldsymbol{\mathcal{SHOIN}(\mathcal{D})}$ **view impact rule, allowing identifying the set of extractions** $\mathcal{S}_{O_i}^{K}$ **composing a view result** $V_{i\mathcal{S}_O}$**, which starter elements** ε_i **are impacted** by the set $\boldsymbol{B_{logi}^-}$ of deletion changes identified by the ontology impact rule. It is used to assess the impact of changes on a view definition. Elements ε_i^- of the set Ω_O^{Imp} are compared to the starter elements ε_i of each extraction $\mathcal{S}_{O_i}^{\varepsilon j}$

composing each view. For each starter element ε_i corresponding to an element of Ω_O^{Imp}, the corresponding extraction $\mathcal{S}_{O_i}^{\varepsilon j}$ is assumed impacted.

Definition 40: $\mathcal{SHOIN}(\mathcal{D})$ View Impact Rule.

- Given Ω_O^{Imp} the set of elements impacted by deletion changes recorded in log_0;
- Given a view result $V_{i\mathcal{S}_O} = \{\mathcal{S}_{O_i}^{C}, \mathcal{S}_{O_i}^{T}, \mathcal{S}_{O_i}^{R}, \mathcal{S}_{O_i}^{A}, \mathcal{S}_{O_i}^{I}, \mathcal{S}_{O_i}^{V}\}$;

The set of impacted extraction results, noted $\mathcal{S}_{O_i}^{imp}$ of $V_{i\mathcal{S}_O}$ is calculated as follow:

- for each $\mathcal{S}_{O_i}^{\varepsilon j} \subseteq V_{i\mathcal{S}_O}$, with a starter element $\varepsilon_j \in \{sC, sT, sR, sA, sI, sV\}$ and $j \in \mathbb{N}$,
  - for each deletion change $\beta_{Bk} \in B_{log\,i}^{-}$, with $k \in \mathbb{N}$, having an element $\varepsilon_k^{-} \in \Omega_O^{Imp}$ as parameter,
    - if the starter element ε_j of $\mathcal{S}_{O_i}^{\varepsilon j_i}$ is equivalent to an element ε_k^{-}, then $\mathcal{S}_{O_i}^{\varepsilon j_i} \in \mathcal{S}_{O_i}^{imp}$;
    - end if;
  - end for;
- end for;

In other words, if the starter element ε_j of at least one extraction result $\mathcal{S}_{O_i}^{\varepsilon j}$ corresponds to an element ε_k^{-} deleted from $\mathcal{S}_O$ as represented in log_0, the set of impacted extractions results $\mathcal{S}_{O_i}^{imp}$ is not null, the view result $V_{i\mathcal{S}_O}$ is impacted and the view definition $V_i(\mathcal{S}_O)$ has to be updated. To resolve this impact, a third rule, called **view update rule**, and a fourth, called **sub-ontology update rule**, are defined in the following parts.

1.3 View Update Rule

This part defines the $\mathcal{SHOIN}(\mathcal{D})$ **view update rule** used to **update the view definition** $V_i(\mathcal{S}_O)$ of a view V_i, if the corresponding set of impacted extractions results $\mathcal{S}_{O_i}^{imp}$ processed by the view impact rule is not null. It simply deletes the view extractions $E(\varepsilon_j)$ corresponding to view results of the set $\mathcal{S}_{O_i}^{imp}$ from the view definition $V_i(\mathcal{S}_O)$.

Definition 41: $\mathcal{SHOIN}(\mathcal{D})$ View Update Rule.

- Given the view definition $V_i(\mathcal{S}_O) = \bigcup_{i=1}^{n} E(\varepsilon_j)_j$ of a view V_i, with the corresponding starter extractions $E(\varepsilon_j)_j$;
- Given the set of impacted extractions $\mathcal{S}_{O_i}^{imp}$;

The update of V_i consists in the deletions of the set E^{imp} of starter extractions $E(\varepsilon_j)_j$ corresponding to the extraction results $\mathcal{S}_{O_i}^{\varepsilon j}$ listed in $\mathcal{S}_{O_i}^{imp}$ such that $V_i(\mathcal{S}_O)^{new} = \{V_i(\mathcal{S}_O)\} - \{E^{imp}\}$.

Note that if $\mathcal{S}_{O_i}^{imp}$ is empty, then $V_i(\mathcal{S}_O)^{new} = V_i(\mathcal{S}_O)$, i.e. the view definition remains the same.

1.4 Sub-Ontology Update Rule

This part defines the $\mathcal{SHOIN}(\mathcal{D})$ **sub-ontology update rule** used to extract the sub-ontology according to a view definition $V_i(\mathcal{S}_O)^{new}$, impacted or not, from the evolved ontology $\mathcal{S}_O{}^{new}$. The sub-ontology update rule explains how to update the sub-ontology $\mathcal{S}_O{}'_i$ defined by a view V_i according to its definition $V_i(\mathcal{S}_O)^{new}$ resulting from the View Update Rule. It shows that the evolution of the sub-ontology $\mathcal{S}_O{}'_i$ into $\mathcal{S}_O{}'^{new}_i$ can be obtained by specifying $V_i(\mathcal{S}_O)^{new}$ on the evolved ontology $\mathcal{S}_O{}^{new}$.

Definition 42: $\mathcal{SHOIN}(\mathcal{D})$ Sub-Ontology Update Rule.

Given a view definition $V_i(\mathcal{S}_O)^{new}$, the update of the corresponding sub-ontology $\mathcal{S}_O{'}_i$ consists in the extraction of a new sub-ontology $\mathcal{S}_O{'}_i^{new}$ from $\mathcal{S}_O{}^{new}$ according to $V_i(\mathcal{S}_O)^{new}$ producing the view result $V_{i\mathcal{S}_O}{}^{new}=\{\mathcal{S}_{O_i}^{C\text{new}}, \mathcal{S}_{O_i}^{T\text{new}}, \mathcal{S}_{O_i}^{R\text{new}}, \mathcal{S}_{O_i}^{A\text{new}}, \mathcal{S}_{O_i}^{I\text{new}}, \mathcal{S}_{O_i}^{V\text{new}}\}$.

In other words, the evolution of a view V_i into V_i^{new} consists in the extraction of a new sub-ontology $\mathcal{S}_O{'}_i^{new}$ from the evolved ontology $\mathcal{S}_O{}^{new}$ according to its view definition $V_i(\mathcal{S}_O)^{new}$ (c.f. Chapter 6 View Extraction Process).

1.5 Discussion

This section has presented the **impact management process** of the **Top-Down Propagation** approach, used in the **change propagation phase** of the ontology evolution process to manage the impact of changes applied on a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views. This approach constitutes the **first type of change propagation** proposed in the OntoVersionGraph methodology. The **second type**, i.e. propagation of changes from a view to its source ontology and the other views defined on it, called the **Bottom-Up Propagation** approach, follows a similar process and reuses the Top-Down Approach to handle the propagation of the source ontology changes to the other views.

2 Bottom-Up Propagation Approach

This section presents the **Bottom-Up Propagation approach** used to **propagate changes from an evolved view to its source $\mathcal{SHOIN}(\mathcal{D})$ ontology and from this ontology to its other views**. It allows managing the **second type of change propagation** of the OntoVersionGraph methodology, i.e. when a change occurring in a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology defined by a view on a $\mathcal{SHOIN}(\mathcal{D})$ ontology, has to be propagated to this ontology. A first part defines the **$\mathcal{SHOIN}(\mathcal{D})$ ontology update rule** used to propagate the changes implemented in the sub-ontology to its source ontology. A second part describes the **evolution process of the ontology**, from the change detection phase to the change propagation phase, in order to assess the logical consistency of the changes applied on the view on the whole ontology set of axioms. It ensures changes propagated to the other views will not cause logical inconsistencies in the corresponding sub-ontologies. A third part describes the **second change propagation process** used to propagate the changes assumed consistent to its other views. It uses the **Top-Down Propagation** approach described in the first section.

The **Bottom-Up Propagation** Approach follows four steps illustrated in Figure 25. It considers the application of changes on a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology $\mathcal{S}_O'$ extracted from a $\mathcal{SHOIN}(\mathcal{D})$ ontology $\mathcal{S}_O$ according to a view V_i.

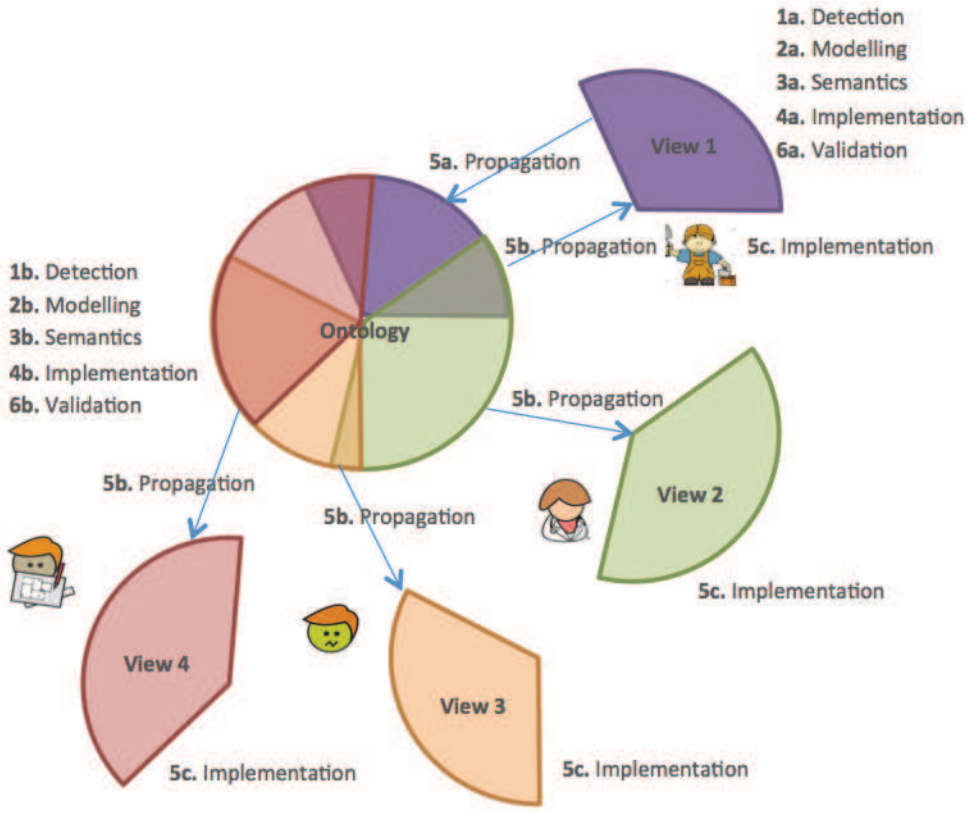


Figure 25. Bottom-Up Approach

First, sub-domain experts launch the **ontology evolution process** of their corresponding sub-ontology $\mathcal{S}_O'_i$ on which changes have to be applied according to the OntoVersionGraph methodology (c.f. 1a.Detection , 2a. Modelling, 3a. Semantics on Fig.25). Once changes are assessed structurally and logically consistent, they are implemented on a copy of $\mathcal{S}_O'$, noted $\mathcal{S}_O'_1$ (c.f. 4a. Implementation on Fig.25). Second, if the second logical consistency check is validated, the **change propagation phase** of the sub-ontology evolution process is launched to **spread the changes applied on $\mathcal{S}_O'$ to its source ontology $\mathcal{S}_O$** (c.f.

4a.Propagation on Fig.25). The set of changes to propagate is processed according to the $\mathcal{SHOIN}(\mathcal{D})$ **ontology update rule** (c.f. Definition 43). Third, $\mathcal{S}_O$ runs a new evolution process until a second change propagation phase (c.f. 1b.Detection, 2b. Modelling, 3b.Semantics on Fig.25). Logical consistency of the whole ontology $\mathcal{S}_O$ is checked according to these propagated changes. If inconsistencies are detected, the **minimal inconsistent sets of changes** are returned to the sub-domain experts responsible for all the sub-ontologies impacted by this inconsistency and of $\mathcal{S}_O$'. They are invited to collaboratively resolve the inconsistency by choosing a **belief change strategy** (c.f. $\mathcal{SHOIN}(\mathcal{D})$ Logical Consistency Preventive Approach in Chapter 5 Section 3). When propagated changes are validated as consistent, they are implemented on a copy of $\mathcal{S}_O$, noted $\mathcal{S}_{O\,1}$, and the second logical consistency check is done (c.f. 4b.Implementation on Fig.25). Four, to **propagate the changes implemented on $\mathcal{S}_O$ to the other views' sub-ontologies**, the **second propagation phase** follows the **Top-Down Propagation** approach to propagate changes from $\mathcal{S}_O$ as described in Section 1 (c.f. 5b.Propagation on Fig.25). The five steps of the Top-Down Propagation approach are completed to extract the sub-ontologies of each view from the ontology copy $\mathcal{S}_{O1}$. It is followed by the ontology validation phase (c.f. 6b.Validation on Fig.25), which replaces $\mathcal{S}_O$ by $\mathcal{S}_{O1}$ becoming $\mathcal{S}_O{}^{new}$.

2.1 $\mathcal{SHOIN}(\mathcal{D})$ Ontology Update Rule

This part defines the $\mathcal{SHOIN}(\mathcal{D})$ **ontology update rule,** used to propagate the changes implemented in the sub-ontology $\mathcal{S}_O$' to its source ontology $\mathcal{S}_O$. As a sub-ontology is defined by a view on an ontology, i.e. is a portion of the latter, any change applied on the sub-ontology impacts the source ontology. Therefore the set of changes to propagate from $\mathcal{S}_O$' to $\mathcal{S}_O$, is the complete set of changes represented in the log of changes of the current evolution of $\mathcal{S}_O$', noted $log_{i'}$. This set is defined by the ontology update rule below:

Definition 43: $\mathcal{SHOIN}(\mathcal{D})$ Ontology Update Rule.

Given $log_{i'}$, the log of changes issued from the semantic phase of the i^{th} ontology evolution of the sub-ontology $\mathcal{S}_O$' extracted from an ontology $\mathcal{S}_O$; the **set of changes to propagate from $\mathcal{S}_O$' to update $\mathcal{S}_O$** is composed of the complete set of changes of $log_{i'}$, represented in log_{j+1} the log of changes of the $j+1^{th}$ evolution process of $\mathcal{S}_O$, such that $log_{j+1} = log_{i'}$.

Once this **first change propagation phase** achieved, a new **ontology evolution process** starts for $\mathcal{S}_O$ according to the changes represented in log_{j+1}.

2.2 Source Ontology Evolution

This part describes the **ontology evolution process of the source ontology $\mathcal{S}_O$** to update, according to the log of changes log_{j+1}.

Including an intermediate ontology evolution process between evolution of the sub-ontology and its change propagation, allows **assessing and resolving the logical consistency of the changes** propagated from the sub-ontology $\mathcal{S}_O$' to the other sub-ontologies $\mathcal{S}_{Oi}$' extracted from $\mathcal{S}_O$, without checking logical consistency for each of them. As logical consistency is assessed on the whole set of axioms of $\mathcal{S}_O$, it ensures that changes later propagated to the other sub-ontologies, will not cause logical inconsistencies. The evolution process of $\mathcal{S}_O$ is launched, until the propagation phase, in order to evaluate the logical consistency of their application on $\mathcal{S}_O$, according to the double check provided by the change semantics and the change implementation phases. Logical inconsistencies should be detected within the first check, i.e. by the **preventive analysis of the change semantics phase**, but, in case, they may be identified in the second check, i.e. by the Pellet reasoner, during the **corrective analysis of the change implementation phase**.

If inconsistencies are identified by the **logical consistency preventive analysis** (c.f. Chapter 5 Section 2), then minimal inconsistent sets of changes, noted *listOfMIC*$_i$, can be returned to the subdomain experts, whose sub-ontologies are impacted by the inconsistencies. In order to **identify the sub-ontologies impacted by these minimal inconsistent sets** of changes, the **view impact rule** used in the **Top-Down Propagation approach** can be applied on the view result of each of the sub-ontologies $\mathcal{S}_{Oi}'$ extracted from $\mathcal{S}_O$. It just needs to take the set of parameters, of each change contained in the minimal inconsistent sets, as the set of impacted elements Ω_O^{Imp}. For each sub-ontology view result $V_{i\mathcal{S}_{Oi}}'$, if the set of impacted extraction results $\mathcal{S}_{Oi}^{imp\prime}$ returned is not null, then the corresponding sub-ontology $\mathcal{S}_{Oi}'$ is impacted. The sub-domain experts in charge of each impacted $\mathcal{S}_{Oi}'$, are therefore noticed of the minimal inconsistent sets of changes detected. They can therefore collaboratively resolve the detected inconsistencies by agreeing on a belief change strategy.

If after changes are implemented on $\mathcal{S}_O$, **Pellet finds inconsistencies**, which were not detected by the first check, then its **conclusions can also be spread** to the corresponding sub-domain experts. As Pellet returns the unsatisfiable axioms causing inconsistencies, the identification of sub-ontologies impacted by these inconsistencies can follow the same process as the one used for the first check. The set of impacted elements Ω_O^{Imp} corresponds then to the ontological elements used in these axioms.

If **no inconsistency is found**, by both preventive and corrective checks, or **when the detected ones are resolved**, **the second change propagation phase** can be launched, to propagate the consistent changes to the other sub-ontologies.

2.3 Source Ontology Changes Propagation

This part describes the second change propagation process used to propagate the changes, propagated by the view V_i defining $\mathcal{S}_O'$ to it source ontology $\mathcal{S}_O$, to the other views.

The **Top-Down Propagation** approach, as defined in the previous section, is used here to **propagate the changes, implemented on the source ontology $\mathcal{S}_O$** and assessed as consistent, to each sub-ontology $\mathcal{S}_O'_i$ extracted from $\mathcal{S}_O$ defined by a view V_i. It starts directly at the second step, i.e. by applying the **ontology impact rule on** the ontology last log of changes log_{j+1}, in order to evaluate the impact of the evolution of $\mathcal{S}_O$ on its ontological elements. According to the Top-Down Propagation approach, it is followed by the **view impact rule** assessing the impact on the view results $V_{i\mathcal{S}_O}$ of each of its views V_i. Then the **view update rule** allows to update their view definitions $V_i(\mathcal{S}_O)$ and the **sub-ontology update rule** re-extract each sub-ontology $\mathcal{S}_O'_i$ defined by each of the views V_i of $\mathcal{S}_O$ from the new copy of $\mathcal{S}_O$, noted $\mathcal{S}_{O1}$.

2.4 Discussion

This section has presented the **impact management process** of the **Bottom-Up Propagation** approach, used in the **change propagation phase** of the ontology evolution process to manage the impact of changes applied on a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology, defined by a view on a $\mathcal{SHOIN}(\mathcal{D})$ ontology, to its source ontology and the other views defined on it. This approach constitutes the **second type of change propagation** proposed in the OntoVersionGraph methodology.

3 Change Impact Management Example

This section illustrates, within two examples, the application of the entire impact management process, according to the two types of change propagation approaches presented in this chapter, i.e. the Top-Down Propagation approach and the Bottom-Up Propagation Approach.

In these examples a knowledge base describes a building and its different sub-domain experts views. The terminological level is a $\mathcal{SHOIN}(\mathcal{D})$ terminology $\mathcal{S}_O$, which defines a standard model of building, whereas the assertional level instantiates the model to described a particular building. Views can be specified on both terminological and assertional levels depending on the business skills of the domain experts. However, an expert of the building model should have a global view on the terminological level, whereas an expert of a sub-domain of a particular building should have a view on the assertional level or on both levels, i.e. the building itself. To reduce the complexity of both examples, only sub-domain expert views on both levels are considered in the two parts of this section. Also to simplify, the use of the term "ontology" will refer to the whole knowledge base. A first part describes the impact management of an assertional change occurring in a sub-ontology defined by a view on the building and inconsistently impacting another view. A second part describes the impact management of a terminological change occurring in the building model defined by the $\mathcal{SHOIN}(\mathcal{D})$ terminology and affecting the whole knowledge base, including the views.

3.1 Assertional Change Example

This first part describes the impact management of an assertional change occurring in a sub-ontology $\mathcal{S}_{O1}'$ defined by an "architect" view V_1 on the building ontology $\mathcal{S}_O$, including terminological and assertional levels. Let us focus on a common issue occurring in the lifecycle of a building: the architect expert wants to represent a hole in an existing wall of the building to set a doorway.

3.1.1 Introduction to the problem

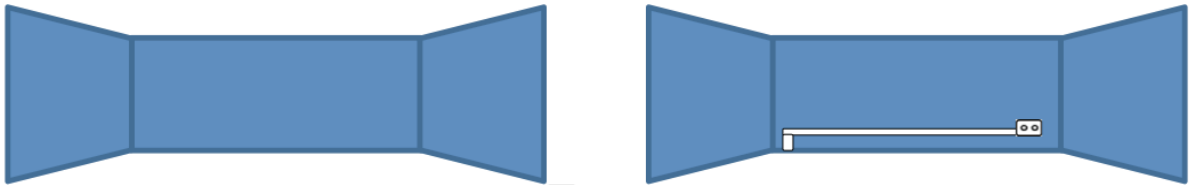


Figure 26. The Architect Point of View on a Wall (on the left) & the Electrician Point of View on the same Wall (on the right)

Let us state that, from the architect point of view, the wall does not contain any architectural constraint, preventing him from setting this doorway (c.f. Fig.26 left view). But from an electrician point of view (c.f. Fig. 26 right view), making a hole in this wall may contradict the electrician view conceptualization as well as the whole knowledge base.

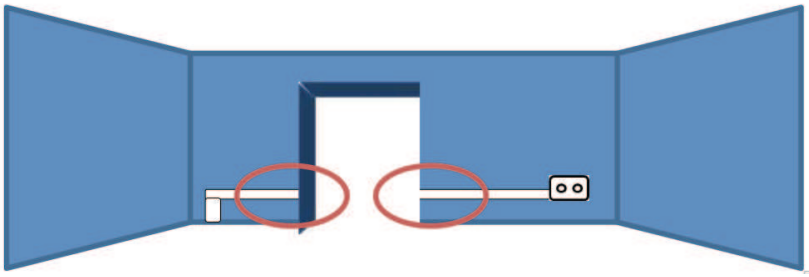


Figure 27. Inconsistent Brute Application of the Modification from the Electrician Point of View

Indeed if an electrical conduit passes along the wall, setting a doorway may cut the conduit. More formally it means that the doorway wall space overlaps with the electrical conduit one (c.f. Fig. 27). Let us imagine it is not permitted by the logical constraints of wall spaces, which can contain maximum one object. To simplify the description of the example, let us model the wall like a grid of twelve wall spaces as illustrated below (c.f. Fig.28):

Figure 28. The Wall as a Grid of Wall Spaces with the Inconsistent Brute Application

The red area is composed of the wall spaces containing the electrical conduit (c.f. wallSpace1, wallSpace2, wallSpace3 and wallSpace4 on Fig.28). The green area delimits the space of the doorway (c.f. wallSpace2 and wallSpace6 on Fig.28). The striped wall space (c.f. wallSpace2 on Fig.28) is the one where the electrical conduit and the doorway overlap. A consistent application of the modification would imply to divert the electrical conduit or to remove it. In practice, several solutions can be chosen: removing the electrical conduit and the outlet, passing it at the top of the doorway, or in the ground (c.f. Fig. 29) or in the ceiling or along another wall.

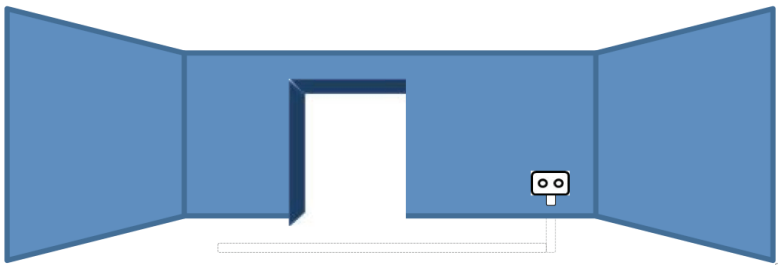


Figure 29. Consistent Application of the Modification from the Electrician Point of View

Let us model this consistent application on the grid of wall spaces (c.f. Fig. 30). As illustrated, the doorway space does not overlap anymore with the electrical conduit one.

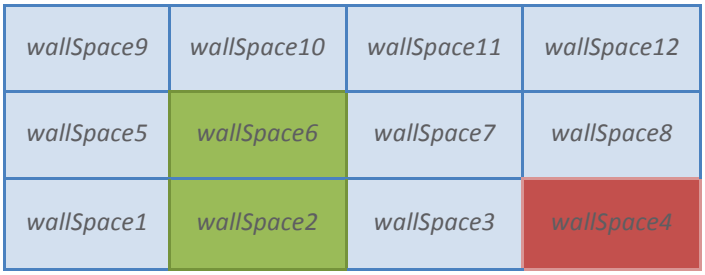


Figure 30. The Wall as a Grid of Wall Spaces with A Consistent Application

This example shows this consistent application of changes can be handled semi-automatically on the whole building knowledge if applied by the architect expert. It explains how to resolve the change impact for other sub-domain experts' views through the two types of automated change propagation approaches, i.e. Bottom-Up and Top-Down Propagation approaches.

3.1.2 Architect Change Management Process: Round 1

This paragraph describes the first round of the change management process launched by the architect expert.

Architect Sub-ontology $\mathcal{S}_{O1}$' Evolution:

- ***Change Detection:*** For the architect expert, who is not aware of the electrical conduit presence on "wallSpace2", the need of change corresponds to set a doorway on the space defined by "*wallSpace2*" and "*wallSpace6*".
- ***Change Modelling:*** From a change modeling point of view, this modification refers to the addition of an instance of the concept "*Doorway*", and the addition of the instantiation of the role "*containsPartOf*" with this instance in the range and the instances of the concept *WallSpace* concerned in the domain ("*wallSpace2*" and "*wallSpace6*"). The corresponding log of changes log_1 can be represented: log_1*=((addConceptInstanciation,(doorway1, Doorway)), (addRoleInstantiation,(containsPartOf, wallspace2, doorway1)), (addRoleInstantiation, (containsPartOf, wallspace6, doorway1)))*. According to the structural consistency constraints, this sequence of changes does not contain any impacting change. There are only additions of basic changes. So the structural consistency is preserved. The log of changes log_1 does not need to be structurally resolved.
- ***Change Semantics:*** As changes represented in log_1 are only basic changes, they do not need to be developed in order to extract the unsatisfiability constraints of their corresponding basic change operator type. The logical consistency preventive approach checks log_1 and does not detect any inconsistency. Indeed the changes chosen by the architect expert does not contradict the knowledge of his own view.
- ***Change Implementation:*** Changes of log_1 assumed as structurally and logically consistent are then applied on a copy of the sub-ontology $\mathcal{S}_{O1}$', *noted* $\mathcal{S}_{O1.1}$'. The second logical consistency check is handled on $\mathcal{S}_{O1.1}$' by the reasoner Pellet. As expected $\mathcal{S}_{O1.1}$' is assessed consistent. Changes can therefore being propagated to the source ontology $\mathcal{S}_O$.
- ***Change Propagation:*** As changes come from the evolution of $\mathcal{S}_{O1}$', which is a sub-ontology extracted by a view V_1 from the ontology $\mathcal{S}_O$, the change propagation type corresponds here to the Bottom-Up Propagation approach. It implies starting a new evolution process for the source ontology $\mathcal{S}_O$ according the changes represented in log_1.

The sub-ontology evolution process is stopped until the new evolution process of the source ontology is validated. Then it will be able to achieve the change validation phase.

Building Ontology $\mathcal{S}_O$ evolution (1st iteration):

3. ***Change Detection and Modelling:*** These phases do not need to intervene as changes are already represented in the log of changes log_1 and have been assessed structurally consistent. Indeed if changes to apply on a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology extracted from a $\mathcal{SHOIN}(\mathcal{D})$ ontology are assessed structurally consistent, then they are also consistent on the ontology. There are two reasons. First, both sub-ontology and ontology respect the $\mathcal{SHOIN}(\mathcal{D})$ ontology model, so changes, which are also compliant with the model, cannot be miswritten. Second, only deletion changes can cause the remaining structural inconsistencies and if a deletion change applied on a subset of the ontology is structurally consistent, i.e. the ontological element to delete exists in the subset, then it is also consistent for the whole ontology, which contains the subset.

- ***Change Semantics :*** To check the consistency of the whole ontology, according to the log of changes log_1, log_1 is concatenated with the global log of changes log_0 of the ontology. Let us state that log_0 contains the role maximal cardinality restriction addition change *(addRoleMaxCardinalityRestriction, (containsPartOf, 1))* and the previous role instantiation addition change *(addRoleInstantiation, (containsPartOf, wallSpace2, electricalConduit1)).* According to the logical unsatisfiabilities of the change operator type *addRoleMaxCardinalityRestriction*, the arity *"1"* of the maximal cardinality restriction of the role *containsPartOf* forbids the instantiation of this role with more than one instance in the range per instance in the domain. However the instance *"wallSpace2"* in the domain of *containsPartOf* has already been used with the instance *"electricalConduit"* in the range. The addition of the instantiation *(addRoleInstantiation, (containsPartOf, wallSpace2, doorway1))* then it does not respect the maximal cardinality constraint. The application of this change is qualified logically inconsistent and has to be resolved by an alternative solution if the architect expert wants to apply the change. The following minimal inconsistent set of changes identified in the global log is given to the architect expert and all the sub-domain experts whose sub-ontology is impacted by this inconsistency: (*addMaxCardinalityProperty, (containsPartOf, 1)), (addRoleInstanciation, (containsPartOf, wallspace2)), (addRoleInstanciation, (containsPartOf, electricalConduit1))*.

The ontology evolution process is stopped until the sub-domain experts propose a new set of changes to apply on the ontology. In addition, the architect sub-ontology evolution process is cancelled, the copy of $\mathcal{S}_{O1}'$, $\mathcal{S}_{O11}'$, is deleted. Also, during the collaborative logical consistency resolution, the architect expert can, for example, ask the electrician expert to proceed to a modification of the electrical conduit path in order to avoid passing *"wallSpace2"* and also *"wallSpace6"*. The current change management process round is interrupted and the current log log_1 is stored until the architect expert is notified by the confirmation of the demanded update. Then he will be invited to pursue this change management process in a second round.

3.1.3 Electrician Change Management Process

This paragraph describes the change management process supervised by the electrician expert in response to the architect expert needs.

- ***Change Detection:*** The sub-domain experts have reached a consensus and the electrician expert has received the modification request of the architect expert. He is aware of the location of the doorway wanted by the architect and can choose another path for the electrical conduit. If he chooses to pass it in the ground then he can remove it from the space defined by "*wallSpace1*", "*wallSpace2*" and

"*wallSpace3*" and adding it in similar ground spaces (not illustrated here to simplify the example presentation). He can model this change on its own sub-ontology $\mathcal{S}_{O2}'$.

- ***Change Modeling:*** This change refers to the deletion of the instantiations of the role *containsPartOf* with the instance "*electricalConduit1*" in the range for the corresponding wall spaces in the domain, and the addition of the similar instantiations of this role but with ground spaces in the domain. The corresponding log of changes log_2 can be modelled:

log_2=((deleteRoleInstantiation, (containsPartOf, wallspace1, electricalConduit1)),
deleteRoleInstantiation, (containsPartOf, wallspace2, electricalConduit1)),
deleteRoleInstantiation, (containsPartOf, wallspace3, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace1, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace2, electricalConduit1),
addRoleInstantiation, (containsPartOf, groundSpace3, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace4, electricalConduit1)).

According to the structural consistency constraints, this set of changes does not contain any impacting change. So the structural consistency is preserved. log_2 does not need to be structurally resolved.

- ***Change Semantics:*** Considering the new path modelled for the electrical conduit does not interfere with other constraints of the electrician expert view, the logical consistency preventive approach detects then no inconsistency, such that log_2 is qualified consistent.
- ***Change Implementation:*** The sequence of changes is then implemented on a copy of the sub-ontology *noted* $\mathcal{S}_{O21}'$. The set of changes of log_2 is concatenated with the ontology global log of changes log_0. Then the second logical consistency check realized by Pellet assesses it as consistent. The change propagation to the source ontology is then launched. Here again, the Bottom-Up Propagation approach is used, reactivating the current ontology evolution process.

Building Ontology $\mathcal{S}_O$ evolution (2nd iteration):

- ***Change Detection and Modelling:*** These phases do not need to intervene as changes are already represented in the log of changes log_2 and have been assessed structurally consistent.
- ***Change Semantics:*** The set of changes of log_2 is concatenated with the ontology global log of changes log_0. Assuming here again, that the new path modelled for the electrical conduit does not interfere with other constraints of the building ontology, the changes, the logical consistency preventive approach detects then no inconsistency and such that log_2 is qualified consistent for the ontology too.
- ***Change Implementation:*** The sequence of changes is then implemented on a copy of the ontology $\mathcal{S}_O$, noted $\mathcal{S}_{O1}$. Pellet assesses the logical consistency of $\mathcal{S}_{O1}$ and detects no inconsistency. Changes are ready to be propagated.
- ***Change Propagation:*** Now changes have been implemented, they have to be propagated to every sub-domain expert views such as the electrician's and architect's ones. The change propagation type corresponds to the Top-Down Propagation Approach. The impact of the changes on $\mathcal{S}_O$ and on the sub-ontologies extracted from $\mathcal{S}_O$ by its different views V_i has to be evaluated, before update each view definition $V_i(\mathcal{S}_O)$ and generating the sub-ontologies new versionsTapez une équation ici.. For each of them the system generates the set of changes B^-_{logi} containing changes of log_2 identified as impacting the view definition. The electrician expert view definition may be impacted as it contains the ontological elements deleted by these changes. The architect view is on the contrary not impacted because his view does not model electrical installations. Then each impacted view definition $V_i(\mathcal{S}_O)$ is updated with the corresponding extractions deletions. Finally all of the sub-ontologies $\mathcal{S}_{O\mathrm{i}}$' are re-extracted from the evolved copy of $\mathcal{S}_O$, $\mathcal{S}_{O1}$, according to their updated or non updated view definitions $V_i(\mathcal{S}_O)$, generating their new versions $\mathcal{S}_{O_i}^{new}$'. The electrician expert sub-ontology $\mathcal{S}_{O2}'$ is

then re-extracted from $\mathcal{S}_O$, according to its updated view definition $V_2(\mathcal{S}_O)$. The architect expert one, $\mathcal{S}_{O2}'$, is also re-extracted even if the corresponding view definition $V_1(\mathcal{S}_O)$ is not impacted.

- ***Change Validation:*** The ontology evolution process is validated and the evolved copy of $\mathcal{S}_O$, $\mathcal{S}_{O1}$, renamed in ${\mathcal{S}_O}^{new}$, replaces $\mathcal{S}_O$, which is removed from the versioning system. log_0 concatenated with log_2 produces the new global log of changes of ${\mathcal{S}_O}^{new}$, noted ${log_0}^{new}$. log_0 is stored in the versioning system as a previous global log of changes of the ontology to be exploited for ulterior versioning purposes. Each sub-domain expert sub-ontology $\mathcal{S}_{Oi}'$ is replaced by their evolved copy ${\mathcal{S}_{Oi}}^{new}'$, such as their corresponding global logs of changes.

3.1.4 Architect Change Management Process: Round 2

This paragraph briefly describes the second round of the change management process of the architect expert cancelled previously. The architect expert has received the update notification from the electrician. He is now assured that he can set the doorway on the space delimited by *"wallSpace2"* and "*wallSpace6*". He can load a new change management process on his evolved sub-ontology ${\mathcal{S}_{O1}}^{new}'$. He reuses the log of changes log_1=*((addConceptInstanciation,(doorway1, Doorway)), (addRoleInstantiation,(containsPartOf, wallspace2, doorway1)), (addRoleInstantiation, (containsPartOf, wallspace6, doorway1)))*, represented during the first change management process (c.f. Architect Change Management Process Round 1) in order to set the doorway. Like in the previous process, the changes are assessed structurally and logically consistent on the sub-ontology. They can be propagated to the source ontology using the Bottom-Up Propagation approach, launching a new ontology evolution process on ${\mathcal{S}_O}^{new}$. As the update of ${\mathcal{S}_O}^{new}$ has permitted to free the two wall spaces required for the doorway from the electrical conduit, setting the doorway does not cause any inconsistency. Changes are therefore assessed as logically consistent on ${\mathcal{S}_O}^{new}$, such that the change propagation to the other views, using the Top-Down Propagation approach can be started. Every sub-ontology ${\mathcal{S}_{Oi}}^{new}'$ defined by a view ${V_i}^{new}$ on ${\mathcal{S}_O}^{new}$ are updated. The sub-ontology ${\mathcal{S}_{O1}}^{new}'$ and the ontology evolution processes ${\mathcal{S}_O}^{new}$ are both validated, producing new versions similarly as the previous electrician change management process (c.f. Electrician Change Management Process).

3.2 Terminological Change

This second part describes the impact management of a terminological change occurring in the building norm, i.e. impacting the terminology of the building knowledge base. Considering the change corresponds to the addition of a new logical constraint, stating that a ground space cannot contain anything, it implies that electrical conduits cannot be buried in the ground anymore. So the solution envisioned in the previous example cannot be applied anymore (c.f. Fig.31).

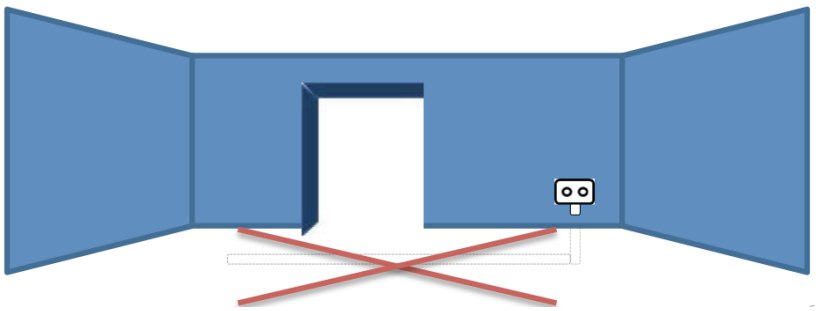


Figure 31. Inconsistent Application of the Modification from the Electrician Point of View w.r.t. the New Model Constraint

Figure 32 shows the ground spaces partially composing the ground of the previous example taking into account the new logical constraint. The striped area (cf. groundSpace 9, groundSpace 10, groundSpace 11 and groundSpace 12 on Fig. 32) models the ground spaces containing the electrical conduit buried in the ground.

groundSpace 18	groundSpace 19	groundSpace 20	groundSpace 21
groundSpace 14	groundSpace 15	groundSpace 16	groundSpace 17
groundSpace 9	groundSpace 10	groundSpace 11	groundSpace 12
groundSpace 5	groundSpace 6	groundSpace 7	groundSpace 8
groundSpace 1	groundSpace 2	groundSpace 3	groundSpace 4

Figure 32. The Ground as a Grid of Ground Spaces with An Inconsistent Application

It also means that any element modelled as contained in a ground space cause a logical inconsistency. So this terminological change impacts the whole knowledge base, i.e. the terminological and the assertional levels. Assertions concerning electrical conduits contained ground have to be removed from the knowledge base. Concerning the previous example, an alternative solution is to pass the electrical conduit along the wall around the doorway like shown in Fig. 33:

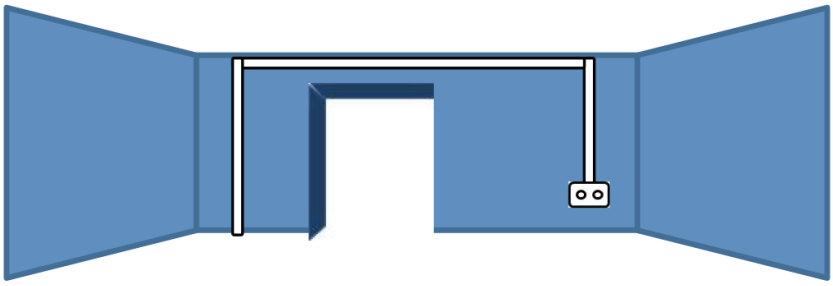


Figure 33. Consistent Application of the Modification from the Electrician Point of View with the New Model Constraint

Let us model this consistent application on the grid of wall spaces (c.f. Fig. 34). As we can see, the doorway space does not overlap with the electrical conduit one.

wallSpace9	wallSpace10	wallSpace11	wallSpace12
wallSpace5	wallSpace6	wallSpace7	wallSpace8
wallSpace1	wallSpace2	wallSpace3	wallSpace4

Figure 34. The Wall as a Grid of Wall Spaces with A Consistent Application with the New Model Constraint

This example shows this consistent application of changes can be handled automatically on the whole building knowledge if applied by the building domain expert. It explains how to resolve the change impact for sub-domain expert views through the two types of automated change propagation approaches, i.e. Bottom-Up and Top-Down Propagation approaches. The change management process launched by the building domain expert on the whole building ontology $\mathcal{S}_O$ can be described as follows.

- ***Change Detection:*** For the building domain expert, who is aware of the norm evolution and has access to the whole building knowledge base, this phase consists in collecting the new constraints identified in the norm. For this example, we consider only one constraint: a ground space must not contain anything.
- ***Change Modelling:*** The corresponding change in the terminology is the deletion of any role signature with a concept as domain and *containsPartOf* as role, the addition of the role universal restriction with the role *containsPartOf* and the bottom concept as range, the addition of the concept intersection with the concept *Space* and the anonymous concept defined by this universal restriction, and the addition of the concept equivalence between *GroundSpace* and this concept intersection. It allows to define the concept *GroundSpace* as a *Space*, which cannot contain a part of anything within a composed change. In the assertional level, this modification implies the deletion of each role instantiation with an individual of the type *GroundSpace* in the domain and *constainsPartOf* as role. The corresponding log of change log_3 can be formalized as below:

log_3=((deleteRoleSignature, (containsPartOf, GroundSpace, Doorway)),
...,
(addComposedConceptEquivalence, (GroundSpace, (addConceptIntersection, (Space,
(addRoleUniversalRestriction, (containsPartOf, BottomConcept)))),
(deleteRoleInstantiation, (containsPartOf, groundSpace2, doorway1)),
...,
(deleteRoleInstantiation, (containsPartOf, groundSpace6, conduit4)))

According to the structural consistency constraints, let us assume this set of changes does not contain any impacting change. log_3 does not need to be structurally resolved.

- ***Change Semantics:*** log_3 is developed in its corresponding basic changes and concatenated to the developed global log of changes log_0 of $\mathcal{S}_O$. Let us consider the logical consistency preventive approach detects no inconsistency such that the application of the changes is qualified consistent.
- ***Change Implementation:*** The set of changes of log_3 is then implemented on a copy of $\mathcal{S}_O$, noted $\mathcal{S}_{O1}$. Assuming Pellet detects no inconsistency, changes can be propagated to the views of $\mathcal{S}_O$.
- ***Change Propagation:*** Now changes have been implemented, they have to be propagated to every sub-domain views updating their corresponding sub-ontologies $\mathcal{S}_{Oi}$'. The change propagation type corresponds to the Top-Down Propagation approach. The impact for each view is analysed. The electrician expert view is impacted as every buried electrical conduits he had modelled have been removed. Considering the architect expert did not model anything buried in a ground space, then his view is not impacted. Then each impacted view definition $V_i(\mathcal{S}_O)$ is updated with the corresponding extractions deletions. Finally all of the sub-ontologies $\mathcal{S}_{Oi}$' are re-extracted from the evolved copy of $\mathcal{S}_O$, $\mathcal{S}_{O1}$, according to their updated or non updated view definitions $V_i(\mathcal{S}_O)$, generating their new versions $\mathcal{S}_{O_i}^{new}$'. The electrician expert sub-ontology $\mathcal{S}_{O2}$' is then re-extracted from $\mathcal{S}_O$, according to its updated view definition $V_2(\mathcal{S}_O)$. The architect expert one, $\mathcal{S}_{O1}$' , is also re-extracted even if the corresponding view definition $V_1(\mathcal{S}_O)$ is not impacted.
- ***Change Validation:*** The ontology evolution process is validated and the evolved copy of $\mathcal{S}_O$, $\mathcal{S}_{O1}$, renamed in $\mathcal{S}_O{}^{new}$, replaces $\mathcal{S}_O$, which is removed from the versioning system. log_0 concatenated with

log_2 produces the new global log of changes of $\mathcal{S}_O{}^{new}$, noted $log_0{}^{new}$. log_0 is stored in the versioning system as a previous global log of changes of the ontology to be exploited for ulterior versioning purposes. Each sub-domain expert sub-ontology $\mathcal{S}_{Oi}{}'$ is replaced by their evolved copy $\mathcal{S}_{Oi}{}^{new'}$, such as their corresponding global logs of changes.

After the building ontology evolution, the electrician expert will need to revise the modeling of the buried electrical conduits, which have all been removed from the knowledge base. According to the consistent resolution illustrated in Fig.30 and Fig. 31, he starts a new change management process on his sub-ontology $\mathcal{S}_{O2}{}^{\text{new}}$' to pass the electrical conduits along the wall. This management process is similar to the previous example one (c.f. Electrician Change Management Process), it uses the Bottom-Up approach to propagate these new changes to the ontology spreading them to the other views if assessed consistent.

3.3 Discussion

These two examples have shown how the **change impact management process** of the OntoVersionGraph methodology can be **collaboratively applied on an evolution problem involving different domain experts** with different views on an ontology. They have explained how to resolve the change impact for sub-domain expert views through the two types of automated change propagation approaches, i.e. Bottom-Up and Top-Down Propagation approaches. As illustrated, changes applied on the ontology have been propagated to the sub-domain views according to the Top-Down Propagation approach. Changes applied on a sub-ontology defined by a view on the ontology have been propagated to the ontology and the other sub-domain views according to the Bottom-Up Propagation approach. These two types of change propagation approaches complete the change propagation phase of the ontology evolution process proposed in the OntoVersionGraph methodology.

4 Conclusion

This chapter has presented the third block of my thesis proposal, i.e. modelling the **impact management process of a consistent change propagation phase** within the OntoVersionGraph evolution process. The impact management process ensures two types of consistent change propagation: first, the propagation from a $\mathcal{SHOIN}(\mathcal{D})$ ontology to its views, called the **Top-Down Propagation** approach, and, second, from a view to its source ontology and the other related views, called the **Bottom-Up Propagation** approach. Both approaches are based on the log of changes used for the evolution of a $\mathcal{SHOIN}(\mathcal{D})$ ontology or a $\mathcal{SHOIN}(\mathcal{D})$ sub-ontology, defined by a view on the ontology, to recover the changes to propagate. They both use the global log of changes of the ontology and the sub-ontologies to assess the impact of one on the other and apply the corresponding subset of changes. Two examples on the field of Building Information Modelling illustrate the whole impact management process within a knowledge base describing a building. The terminological level corresponds to the building model whereas the assertional level describes a particular building. Different sub-domain expert views are considered. The first example describes the impact management of an assertional change occurring in a sub-domain expert view and impacting inconsistently another view. The second example describes the impact management of a change occurred in the building norm impacting the whole knowledge base. For both example, different use of the Top-Down and Bottom-Up approaches are explained.

Thanks to this chapter, the three blocks of my thesis proposal, **extending OCM features to management of ontology views and change propagation between them and their source ontology**, are now completely formalized. The OntoVersionGraph methodology therefore satisfies the three objectives of my thesis: (1) **enable change management of formal ontologies since their creation**, (2) **enable adaptation of formal ontologies to new user assumptions,** and (3) **the coherent change propagation between a formal ontology and its different assumptions.**

Chapter 8

Prototype Implementation

Summary

This chapter describes the **first prototype implementation** of the OntoVersionGraph methodology. A first section gives a general overview on **OntoVersioning**, the Java API prototype dedicated to OWL DL OCM. A second section details the **change management module** implementing the block (1) of the methodology in the API. A third section describes the **view specification and sub-ontology extraction module** implementing the block (2) in the API. A fourth section presents the **change impact management module** implementing the block (3) in the API. These three last sections also bring a discussion about the benchmarks and tests realized for each module to identify the limits of the prototype and the improvements required to go beyond.

Plan

1 OntoVersioning API: Presentation of the Prototype 200
1.1 API Main Features 200
1.1.1 Ontology Change Management Module 200
1.1.2 Ontology View Specification & Sub-Ontology Extraction Module 203
1.1.3 Change Impact Management Module 203
1.2 VersionGraph Content 204
1.3 Other API Modules 206
1.3.1 Project Creation Module 206
1.3.2 Project Modification Module 207
1.3.3 OntoVersioning Project Propagation 208
1.4 Discussion 208
2 Using OWL DL Ontology Change Management in OntoVersioning 209
2.1 Choosing an Operation Verification Mode 209
2.2 Modelling Changes 209
2.3 Implementing and Validating Changes 212
2.4 Discussion 213
3 Specifying Views & Extracting Sub-Ontologies in OntoVersioning 215
3.1 Defining a View 215
3.2 View Extraction 215
3.3 Discussion 216
4 Using Change Impact Management Module in OntoVersioning 220
4.1 Using the Top-Down Propagation Approach 220
4.2 Using the Bottom-Up Propagation Approach 221
4.3 Discussion 222
5 Conclusion and Future Development Perspectives 223

Previous chapters have presented the proposal of my thesis, i.e. **a change management methodology dedicate to formal ontologies, extending OCM features to management of ontology views and change propagation between them and their source ontology,** called OntoVersionGraph. This chapter describes the first attempt to implement this methodology in a **prototype** called **OntoVersioning**. The prototype follows two goals. It must first technically **prove the feasibility of the methodology** by practically assessing the contribution on existing ontologies. The second goal is to provide a **solution easy to integrate into any system** using formal ontologies.

To reach the first goal, the prototype should embed features satisfying the three objectives of my thesis, i.e. (1) enable change management of formal ontologies since their creation, (2) enable adaptation of formal ontologies to new user assumptions, and (3) the coherent change propagation between a formal ontology and its different assumptions. To satisfy (1), it should provide to domain experts an **ontology creation and evolution module** allowing them to develop $\mathcal{SHOIN}(\mathcal{D})$ formal ontologies or update existing ones according to a list of compliant change operators. A semi-automatic evolution process based on the six phases described in the OntoVersionGraph methodology should guide them to ensure a consistent application of changes on such ontologies. In this perspective, the list of changes should be restricted to the $\mathcal{SHOIN}(\mathcal{D})$ semantics. However, until now, it is quite hard to find existing formal ontologies represented in the $\mathcal{SHOIN}(\mathcal{D})$ DL syntax to test their update. Also, OWL DL having the same expressivity level as $\mathcal{SHOIN}(\mathcal{D})$ and OWL DL ontologies being commonly developed and accessible on the Web, I decided to slightly adapt the methodology to offer a **prototype dedicated to OWL DL ontologies**. To satisfy (2), the prototype should offer a **module for OWL DL view specification** to allow domain experts easily defining views on existing ontologies and extracting the corresponding OWL DL sub-ontologies. To satisfy (3), the prototype should provide a **module for change impact management** domain experts should be able to use, when they apply changes on an ontology or a sub-ontology.

To reach the second goal, the prototype should be **portable to any operating system** and **extendable to additional ontology management features**, like visualization tools, collaborative environments etc. This goal has motivated the choice of designing an **API** (Application Programming Interface) extending a commonly used one: the Java framework called **Jena**[15]. An API provides a certain level of abstraction for the developer, who wishes to integrate it in an existing system. Indeed, it hides the complexity of access to the code by providing a standard set of functions, for which only parameters and return values are known. It is then easy to integrate in an existing system. Moreover, Jena is an open source Semantic Web framework in Java used for **building and managing ontologies**. The prototype extending Jena, can therefore take advantage of the ontology management features provided by this framework. Its API allows extracting data from and writing to RDF graphs, including OWL ontologies serialized in the RDF syntax. An RDF graph is represented as an abstract "model", which can be sourced with data from files, databases, IRIs or a combination of these, and also be queried through SPARQL[16] and updated through SPARUL[17]. Jena is similar to the Sesame[18] framework; though, unlike Sesame, Jena **provides support for OWL**. It also embeds various internal reasoners, which can be set up, including the Pellet[19] reasoner. As stated in Chapter 4, **Pellet** is an open source Java reasoner dedicated to OWL DL ontologies. Thanks to Pellet, the prototype will natively be able to **check the logical consistency of OWL DL** ontologies after changes have been implemented on them. According to these requirements, the OntoVersioning prototype has been developed with Java 1.7 and includes all required libraries. To integrate it in a project, a Java Virtual Machine and a Java Development Kit, both up-to-date are sufficient.

[15] *Jena framework : https://jena.apache.org/*
[16] *SPARQL: http://www.w3.org/TR/rdf-sparql-query/*
[17] *SPARUL (SPARQL-Update) : http://www.w3.org/Submission/SPARQL-Update/*
[18] *Sesame framework: http://www.openrdf.org/*
[19] *Pellet reasoner : http://clarkparsia.com/pellet*

A developer can therefore load the jar file containing the API in any Integrated Development Environment (IDE) suited for Java, such as Netbeans[20] or Eclipse[21], to integrate it to an existing project.

This chapter is articulated in fifth sections. The first section gives a general overview on the OntoVersioning API. The second section details the first module displaying the evolution and versioning features in the API. The third section describes the second module related to the specification of views and extraction of sub-ontologies from an ontology in the API. The fourth section presents the third module dealing with the change impact management between an ontology and its views within the API. The fifth section consists in a discussion about the benchmarks and tests realized for each module and identifies the limits of the prototype and the improvements required to go beyond.

1 OntoVersioning API: Presentation of the Prototype

This section gives a general overview on the API dedicated to this approach, called OntoVersioning. A first part describes its features, including ontology evolution and versioning, specification of views and sub-ontology extractions from an ontology, and change impact management between the ontology and its views. A second part describes the composition of VersionGraph files containing the global logs of changes of ontologies and sub-ontologies that are used as support for versioning, consistency maintenance and change propagation. A third part presents the different other modules of the API including creation, loading, edition, saving and propagation of a project.

1.1 API Main Features

The OntoVersioning API consists of several modules, which are all designed to meet a specific feature in the ontology change management system. This part presents the three features implemented in the three modules identified to achieve the objective of my thesis.

1.1.1 Ontology Change Management Module

OWL DL ontology change management, composed of evolution and versioning features, is the first main feature of this prototype.

Figure 35 illustrates a simplified versioning process of the OntoVersioning incremental versioning system based on an ontology evolution process. In this figure, an ontology version V.0 is evolved by the addition of several axioms producing a version V.1, which is also updated producing a version V.2. Each axioms set addition preceded by a "+" on the figure corresponds to a set of changes applied on the ontology, i.e. a log of changes. The whole set of changes delimited by a blue border for each ontology version, corresponds to the global log of changes of the ontology. To cope with the approach formalized in Chapter 4, logs of changes are generated at each ontology evolution and allow building the ontology global log of changes tracing all the changes occurred since the ontology creation. When domain experts validate an evolution, the API replaces the current ontology by its evolved copy version. The global log of changes of the replaced ontology is however kept as a previous log version for versioning purposes.

[20] *Netbeans IDE : https://netbeans.org/*
[21] *Eclipse IDE : https://www.eclipse.org/*

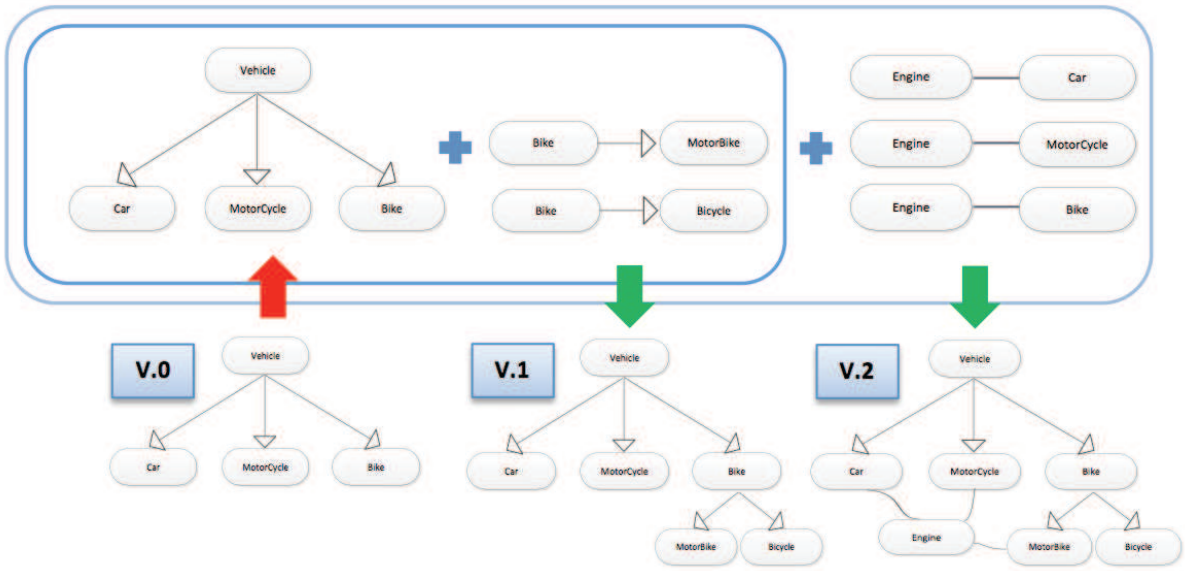


Figure 35. OntoVersioning Simplified Versioning Process

The change management process proposed in OntoVersionGraph is articulated over the six phases of the ontology evolution process (c.f. Chapter 4), which, depending on the satisfaction of structural an logical consistency constraints, are not systematically validated to generate a new ontology version. It is also extended with the specification and the impact management of ontology views. Therefore the versioning process of OntoVersioning is actually a bit more complex than the previously described one. Figure 36 illustrates this extended versioning system.

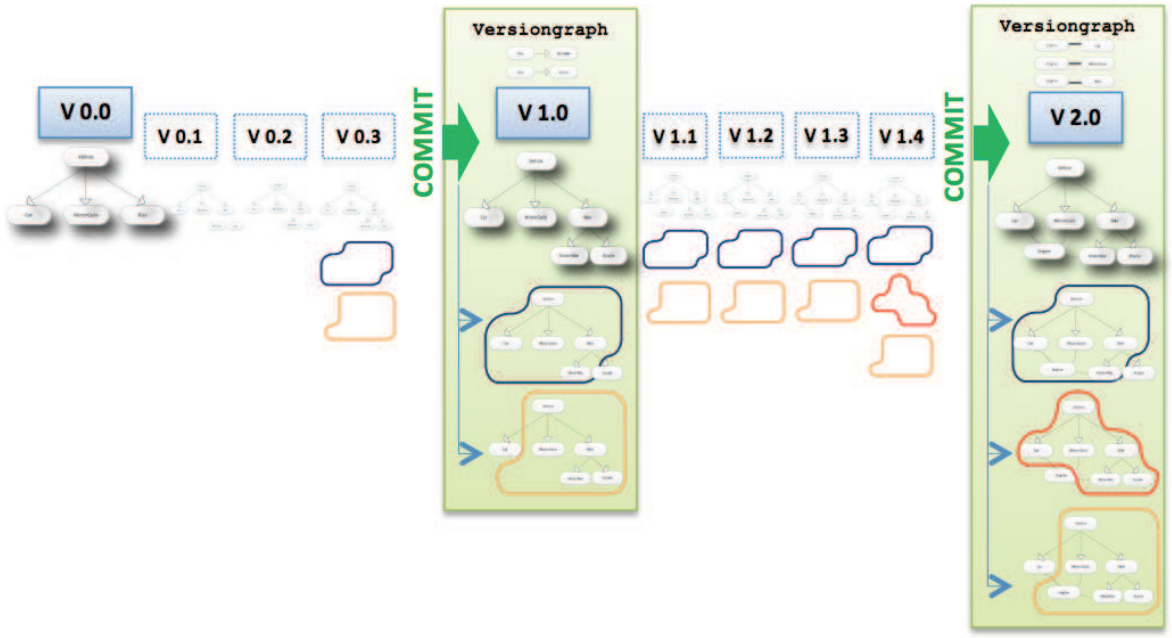


Figure 36. OntoVersioning Extended Versioning Process

According to Figure 36, an initial OWL DL ontology is considered at the V0.0 state. It constitutes the basis for all change applications evolving it into new version and for extracting views. The ontology may be empty or may be an existing one imported in a new OntoVersioning project. If experts of the domain have detected and agreed on changes to apply on the V0.0 ontology, they run a new evolution process to represent them (c.f. Change Modelling phase in Chapter 4), assess the consistency of their

application (c.f. Change Modelling and Change Semantic phases in Chapter 4), propagate them to the potential ontology views or source ontologies (c.f. Change Propagation phase in Chapter 4), implement them and assess the ontology logical consistency with the Pellet reasoner (c.f. Change Implementation phase in Chapter 4), and validate them (c.f. Change Validation phase in Chapter 4). If changes are not assessed as consistent or if the domain experts are not satisfied with the result of the change application on the copy V0.1 of the current ontology V0.0, they can modify it by running a new evolution process until they obtain the expected result and validate it. The validation of the evolution process (c.f. Commit on Fig.33) generates the ontology new global log of changes. Logs of changes and global logs of changes are represented in OWL graphs called VersionGraphs. VersionGraphs are specified in OWL such that they can be queried by SPARQL queries to retrieve a ontology version according to specific changes including view specifications. Indeed, if views have been specified on the ontology, like on its V0.3 evolution (c.f. blue and orange shapes under V0.3 in Fug.33), then the ontology global log of changes represented in a VersionGraph also keeps trace of it. When validated, a new version V1.0 of this ontology is generated, by replacing the initial one by the satisfying evolved copy V0.3. The versioning system stores the new ontology VersionGraph, with the initial one, if the initial ontology already had one. The ontology version V1.0 can then be put into production on a server, without the need of additional libraries or special systems. For the end user of the domain, the access of the ontology by the OntoVersioning API does not change his habits from Jena, as it is totally transparent. The versioning process remains the same for further ontology evolutions (c.f. from V1.0 to V2.0 in Fig.33).

During the change modelling and the change semantics phases of the ontology evolution process, the change management process uses two other modules: the structural and the logical consistency preventive resolution modules.

Like in the OntoVersionGraph methodology, structural consistency is almost guaranteed in the OntoVersioning API by the display of a limited list of operations derived from OWL DL constructs. The change modelling phase however requires a structural assessment for two problems. First, if the IRI typed by the domain expert, to reference a subject, a predicate or an object, as a parameter in a change refers to a non existing resource. Second, if the domain expert calls a deletion change on a class, an object property, a datatype property, a datatype, an individual or a datavalue, which is already used by axioms of the ontology. This second case is resolved by an algorithm inspired by the structural consistency preventive approach formalized in Chapter 5 and adapted to OWL DL change operator type structural constraints.

Logical consistency is also assessed by a preventive approach during the change semantic phase. It is inspired by the logical consistency preventive resolution approach formalized in Chapter 5. However the approach used to identify minimal inconsistent sets of changes in an ontology global log of changes is slightly different. It is also based on the use of unsatisfiability constraints related to the basic change operator types displayed by the language considered, but uses a model checker to parse the log as a graph of states. This technique called CLOCk (Change Log Ontology Checking) presented in (Gueffaz & al, 2012) was implemented as part of another project of the Checksem team called ScaleSem, which focuses on formal approaches for semantic qualification. CLOCk takes as input a VersionGraph global log of changes, translated as a NμSMV graph of states (Gueffaz & al, 2011), and checks it according to OWL DL change operator unsatisfiability constraints, represented by inconsistency patterns, to identify the potential minimal inconsistent sets of changes. It uses the NμSMV model checker to verify the existence of the inconsistency patterns in the graph. Until now, only a few incomplete inconsistency patterns have been formalized in order to assess the feasibility of the technique on a toy example. The logical consistency preventive approach formalization presented in Chapter 5 has improved this approach with a complete range of $\mathcal{SHOIN}(\mathcal{D})$ change operator types and does not impose a particular implementation. Based on the corresponding unsatisfiability constraints, the inconsistency identification algorithm can as well been implemented by checking a VersionGraph global log of changes with a model checker, as well as querying

it with SPARQL queries. Indeed, a VersionGraph being specified in OWL, it is easily interrogated by SPARQL queries.

1.1.2 Ontology View Specification & Sub-Ontology Extraction Module

The second main feature is the specification of views and the extraction of the corresponding OWL DL sub-ontologies from an OWL DL ontology.

Domain experts can agree on the specification of ontology views according to a domain user assumption by only specifying the key elements required for his needs. This semi-automated approach allows easily and rapidly extracting the subset of axioms composing the view result without having to exhaustively specify them. Like in the OntoVersionGraph methodology formalization, defining a view is handled by the representation of a composed change called *extractSubOntology*(e_1, ..., e_n), but parameters e_i are the IRIs of the starter elements (c.f. View Definition in Chapter 6). Also, these starter elements are limited to IRIs of classes, object properties, datatype properties and individuals, whereas the methodology also includes datatypes and datavalues. The set of axioms generated by the application of this change corresponds to the axiomatic entourage of these starter elements, as defined by the four corresponding extraction rules of the methodology (c.f. Starter Extraction Rules in Chapter 6).

The extraction process first retrieves the resources corresponding to the IRIs. It chooses the right starter extraction rule algorithm corresponding to each resource type. It then uses the corresponding complete extraction rule selecting all the axioms defining or using the resource and uses the partial extraction rules to select the ontological elements composing these axioms, such as classes, datatypes, object properties, datatype properties, individuals and datavalues. All these axiom extractions are processed using SPARQL queries translated from the different complete and partial extraction rules defined in the methodology (c.f. Complete and Partial Extraction Rules in Chapter 6). The set of axioms extracted, which is a subset of the source ontology, i.e. a sub-ontology, is then versioned by the versioning system as any OWL DL ontology. It has its own VersionGraph, including its global log of changes, and its extraction is represented in the source ontology VersionGraph global log of changes by the corresponding composed change.

The view extraction process generates the global log of changes of this sub-ontology as a subset of the ontology global log of changes. It is derived from it according to the subset of the ontology axioms defined by the view result. Changes of the ontology global log of changes corresponding to addition operations of axioms contained in the sub-ontology are identified and added to the sub-ontology global log of changes.

1.1.3 Change Impact Management Module

The third main feature is the change impact management between an OWL DL ontology and the sub-ontologies defined by its views, when one of them evolves. The two propagation approaches presented in Chapter 7 are implemented in OntoVersioning.

The Top-Down Propagation approach is part the change propagation phase of an ontology evolution process. Before domain experts validate an evolution iteration of an ontology, changes are propagated to the sub-ontologies defined by its views. The log of changes resulting from this evolution process is used to assess the impact of these changes on each view. If the view definition is impacted then its definition is automatically updated. Then, even if the view definition was not impacted, a new extraction process of the corresponding sub-ontology is launched on the evolved ontology. The sub-ontology VersionGraph is also updated during this new extraction process.

The Bottom-Up Propagation approach is part of the change propagation phase of a sub-ontology evolution process. If a sub-domain expert needs to apply changes on the ontology from its own view, launches the corresponding evolution process. Before he can validate it, these changes are propagated to the source ontology in order to assess the logical consistency of their application on the whole ontology. The log of changes resulting from the sub-ontology evolution is used to put all its changes as input of the new evolution process of the ontology. During the change semantic phase of this process, if changes contained by the log turning the whole ontology inconsistent are detected, then the ontology evolution is cancelled. As for consistency assessment the changes represented in the input log is concatenated with the ontology global log of changes represented in its VersionGraph, the minimal inconsistent set of changes are detected among the whole set of changes. The sub-domain expert responsible for the sub-ontology evolution causing the inconsistency is notified of these inconsistent sets. According to the methodology, the sub-domain experts whose sub-ontologies are also impacted should be also notified, however the current API version does not yet provide the collaborative environment to support this feature. Also when the domain expert finally models changes compliant with the logical constraints of the whole ontology, both sub-ontology and source ontology evolutions are validated and the changes are propagated to the sub-ontologies defined by the other views on the source ontology. Each sub-ontology is re-extracted according to the update their view definition from the source ontology. Their VersionGraphs are all updated according to this update.

1.2 VersionGraph Content

This part describes the content of a VersionGraph storing an ontology or a sub-ontology global log of changes.

The three modules presented in the previous part are all based on the use of logs of changes and global logs of changes, represented in VersionGraphs. Logs are the backbone of the entire change management system of the OntoVersioning prototype, like in the OntoVersionGraph methodology. They support the ontology versioning process, the view specification and sub-ontology extraction process, and the change impact management between an ontology and its views. Representing all the changes applied on an ontology since its creation in a crescent order, a global log of changes allows to chronologically regenerate an ontology version. Regeneration of a previous ontology version can for instance be stopped at a wanted change of the list in order to start a new evolution from this state. It is also used as input of the structural consistency preventive approach during the change modelling phase of the ontology evolution process, and of the logical consistency preventive approach during the semantic phase.

The VersionGraph of an ontology or a sub-ontology is an OWL file. This file contains a light OWL ontology, which is divided into two parts : the history of changes applied on the ontology or the sub-ontology since its creation, i.e. its global log of changes, including the meta model defining them, and contextual data. The history of changes is represented as a chained list of changes, describing the succession of the operations applied to the ontology or the sub-ontology. Contextual data may contain additional information about the ontology or the sub-ontology, specifying for instance the version number, the domain experts responsible for the changes, the name of the ontology, etc.

The main concept of a VersionGraph is the concept *Operation*. It describes all the OWL DL change operator types that can be traced in the history of changes contained by a VersionGraph. Each instantiation of *Operation* represents a single change applied on a single OWL DL ontology, and contains all the elements needed to call and apply the desired change operator type on the required elements of the ontology. The name of each change operator type concept subsumed by Operation is of the form *"[Add|Delete]Operation*" (e.g. *AddClass,* c.f. *addConcept* $\mathcal{SHOIN}(\mathcal{D})$ basic change*, or DeleteObjectProperty*, c.f. *deleteRole* $\mathcal{SHOIN}(\mathcal{D})$ basic change, etc.). Therefore, from its name, we can determine if the change

operator type adds or deletes an axiom of the ontology, and from which OWL DL construct subset is issued this axiom. A change, instantiating *Operation*, takes at most three parameters, called subject, predicate and object (or objects), all typed in strings. A VersionGraph being serialized in RDF/XML, it appears that the *string* type is the most appropriate. The differentiation between a parameter of type "object" and a parameter of type "objects" is due to the need to sometimes specify collections of IRIs as one object. For instance, in the case of the instantiation of the change operator type *AddAllDifferent (c.f. addDifferentInstances* $\mathcal{SHOIN}(\mathcal{D})$ composed change), adding a individual differentiation constraint between several individuals, the parameter has to be a list of individuals' IRIs. Also, "objects" and "object" are mutually exclusive parameters types, they cannot be both used in the same change. Each change is identified by a number, which is unique and is used to indicate its occurrence in all the ontology versions. It can also be linked to others in some cases:

- If a change occurs before it, that is to say, if it is not the first change on the ontology,
- If a change occurs after it, that is to say, it is not the last change,
- If the change is a composed one, i.e. composed of other basic changes,
- If the change is a basic change composing a composed one.

Links are used during the VersionGraph serialization, during the application of the changes on the ontology or during the verification of the existence of a change.

The *Context* concept describes the second part of an ontology or sub-ontology VersionGraph. It describes all data that are not directly related to the history of changes. Some metadata can natively annotate a VersionGraph like its number or the ontology textual description. These metadata will be applied to the ontology properties, like "owl:version" or "owl:description" during an evolution iteration.

Figure 37 below illustrates the Operation object model of the corresponding Java class included in OntoVersioning to manage each change recorded in a VersionGraph.

Figure 37. Concept Operation Object Model

1.3 Other API Modules

This part details the different other modules composing the OntoVersioning API, such as the project creation, the project modification and the project propagation, and explains how to use it. Here is considered a correctly configured IDE with a JVM and JDK up-to-date.

1.3.1 Project Creation Module

An OntoVersioning project enables to develop an OWL DL ontology or sub-ontology from the beginning, but can also take over an existing OWL DL ontology in order to evolve it. In OntoVersioning, a project is a folder that contains the VersionGraphs associated with the ontology versions and its corresponding views, and the last version of the ontology and the last version of the sub-ontologies defined by its views. Each

project focuses on a unique main ontology, and therefore a unique namespace. The namespace is expected to be non-existent at the time of the creation of the project, for there are no conflicts.

If a domain expert decides to develop an ontology with OntoVersioning, the first process will generate a first minimal VersionGraph that will not contain anything in the history of changes, but contextual data. To create a project, there is no need having an existing ontology folder. Indeed, the system will automatically create one if it does not already exist. Also, it is advisable not to select existing folders. The folder name will be the name of ontology, and the namespace is represented in the first VersionGraph metadata, which corresponds to the base of the project. This VersionGraph, which is almost empty, apart from some requested metadata, does not contain any change and therefore symbolizes the empty ontology. For instance, the instruction below creates a project named "test" in the "ontologies" folder, which namespace is "http://example.org/# ".

```
Project p = new Project("ontologies", "test", "http://example.org/#")
```

Taking over an existing OWL DL ontology, which has not been developed inside the API, into a new project requires more effort. Indeed, it implies generating a VersionGraph containing the hypothetical succession of changes that may have been applied to develop the existing ontology. This feature is not implemented in the actual version of the API. However it is expected in a future version. A draft algorithm generating the VersionGraph of an existing OWL DL ontology would involve the following steps:

- Representing the creation of all the classes and their description (c.f. concept in $\mathcal{SHOIN}(\mathcal{D})$),
- Representing the global class subsumption hierarchy description (c.f. concept subsumption in $\mathcal{SHOIN}(\mathcal{D})$),
- Representing the creation of all object properties and datatype properties and their description (c.f. roles and attributes in $\mathcal{SHOIN}(\mathcal{D})$),
- Representing the global object property and datatype property subsumption hierarchy description (c.f. role and attribute subsumption in $\mathcal{SHOIN}(\mathcal{D})$),
- Representing the object property and datatype property signatures description (c.f. role and attribute signatures in $\mathcal{SHOIN}(\mathcal{D})$),
- Representing the function axioms associated with every classes, datatypes, object properties, datatype properties, individuals and datavalues (c.f. concept, datatype, role, attribute, instance and datavalue functions in $\mathcal{SHOIN}(\mathcal{D})$),

1.3.2 Project Modification Module

To open an existing project in OntoVersioning, the system only requires the project path. This path can be relative or absolute. For instance, this instruction opens the project "test" of the folder "ontologies".

```
Project p = new Project("ontologies", "test");
```

Once the project is loaded, it is ready to use. This just requires loading the corresponding VersionGraph like below:

```
VersionGraph vg = p.loadLastVersion();
```

For both ontology development types, i.e. from an empty ontology or an existing OWL DL ontology, it is possible to import an ontology updated by the API, which is already online, by specifying its namespace. In this case the online version of the ontology is loaded. Indeed, one of the main interests of ontologies is to simply bind them, so that it can be reused. For instance, the instructions below create a

project named “test” in a folder “ontologies”, load the corresponding VersionGraph and add to the latter the ontology IRI “iriOfOntologyToImport”.

```
Project p = new Project("ontologies", "test", "http://example.org/#");
VersionGraph vg = p.loadLastVersion();
vg.getOperations().addOntologyImports("iriOfOntologyToImport");
```

1.3.3 OntoVersioning Project Propagation

After several ontology evolutions, the domain experts can decide to validate the whole changes and apply them definitively to generate a new OWL DL ontology version. The API provides a “commit” feature, which replaces the previous ontology by its evolved copy but keeps trace of its VersionGraph version in the versioning system. The API only generates ontology versions in the corresponding folder. Their deployment in the right location has to be done by an administrator.

1.4 Discussion

This section has presented a general overview on the OntoVersioning API features. The main modules are the change management module using the structural and logical preventive resolution modules, the view specification and sub-ontology extraction module, and the change impact management module. These modules are all supported by VersionGraph files containing the global logs of changes of ontologies and sub-ontologies. A VersionGraph being represented as an OWL light ontology describing the whole changes applied on an ontology or a sub-ontology and their related metadata, can be queried by SPARQL queries for many purposes, such as logical inconsistency identification, ontology version retrieving and regeneration, etc. Three different other modules related to the creation, the modification and the propagation of an OntoVersioning project, have been presented. The development and the evolution of an OWL DL ontology are until now realised from scratch but the API is also envisioned to support existing OWL DL ontologies upon the automated construction of its VersionGraph.

2 Using OWL DL Ontology Change Management in OntoVersioning

This section details the use of the change management features in the OntoVersioning API. The first part explains how to choose the verification mode for ontology changes. A second part presents the list of OWL DL change operator types displayed by the API. It explains how to select one of them, how to instantiate it as a change with the right parameters and how to add the change to the ontology log of changes. A third part shows how to implement, save the changes and how to generate of a new version of the ontology. A fourth part presents the results of the tests realized on this module.

2.1 Choosing an Operation Verification Mode

When starting a change management process, the first step is the choice of the change verification mode. This part explains how to choose between the two verification modes proposed by the API. The first is based on the Jena API and requires a certain amount of time and resources. The second is based on the OWL DL structural consistency constraints and also embeds CLOCk, as a preventive logical consistency identification approach. This second mode highly increases the execution performances but, as explained in previous section, still remains at an experimental state and needs thorough tests.

The first instruction below allows using Jena's verification mode.

```
Config.moteurVerification = Config.typeVerifications.WithJena;
```

The second instruction below allows using the OntoVersioning API's verification mode.

```
Config.moteurVerification = Config.typeVerifications.WithoutJena;
```

When the verification mode is chosen, then changes can be modelled.

2.2 Modelling Changes

This part presents the list of OWL DL change operator types displayed by the API. It explains how to select one of them for change modelling, how to instantiate it as a change with the right parameters and how to add the change to a VersionGraph factory, i.e. the log of changes.

The list of basic addition and deletion OWL DL change operators, derived from the changes subset of the OWL DL constructs is presented in Table 11. The first column lists a subset of the complete OWL DL constructs, in which constructs are directly involved in ontological description. It shows for each OWL DL ontological construct the corresponding change operator types and their parameter types. OWL DL constructs are less numerous than the $\mathcal{SHOIN}(\mathcal{D})$ constructs presented in Chapter 4, for several reasons:

– OWL is an ontology language not a knowledge base language, so it namely provides terminological constructs. Some assertional constructs are yet supported by OWL-Lite: individual type such as *rdf:type*, and nominal functions such as *owl:differentFrom* (c.f. instance differentiation in $\mathcal{SHOIN}(\mathcal{D})$) and *owl:samesAs* (c.f. instance equivalence in $\mathcal{SHOIN}(\mathcal{D})$). Also OWL DL, which is based on the $\mathcal{SHOIN}(\mathcal{D})$ DL semantics, contains the additional assertional constructs: *owl:oneOf* (c.f. instance enumeration in $\mathcal{SHOIN}(\mathcal{D})$) and *owl:hasValue* (c.f. role value restriction in $\mathcal{SHOIN}(\mathcal{D})$). However, $\mathcal{SHOIN}(\mathcal{D})$ assertional constructs such as datatype, role and attribute instantiations, respectively defining datavalues, role and attribute assertions, are not provided by specific OWL DL constructs. The corresponding role and attribute assertions can however be represented in OWL DL by a single

RDF triple. For instance in the OWL piece of code below, an individual named *individual1*, typed by a class *Class1*, is the subject of an object property assertion represented by a triple taking *objectProperty1* as predicate and the individual named *individual2* as object.

```
<owl:NamedIndividual rdf:about="http://www.site.org/o.owl#individual1">
    <rdf:type rdf:resource="&example;Class1"/>
    <example:objectProperty1 rdf:resource="http://www.site.org/o.owl#individual2"/>
 </owl:NamedIndividual>
```

Therefore, to allow adding or deleting such assertions, the corresponding change operator types need to be added to the list of basic changes derived from OWL DL constructs.

– The datatypes that can be used in OWL are issued from the XML Schema datatypes[22] (XSD) list. The XSD list is considered as a fixed list for an OWL ontology, it is not described by the ontology. Indeed it is only extendable out of the ontology description. Therefore no addition or deletion of datatypes in an OWL ontology is considered.

OWL DL Constructs	***Change Operator Types***	***IRI Parameters Type***	
		Subject	***Object***
owl:allValuesFrom	*AddAllValuesFrom* *DeleteAllValuesFrom*	*ObjectProperty* */DatatypeProperty*	*Class*
owl:cardinality	*AddCardinalityProperty* *DeleteCardinalityProperty*	*ObjectProperty*	*xsd:integer*
owl:Class	*AddClass* *DeleteClass*	*Class*	
owl:complementOf	*AddComplementOf* *DeleteComplementOf*	*Class*	*Class*
owl:DatatypeProperty	*AddDatatypeProperty* *DeleteDatatypeProperty*	*DatatypeProperty*	
owl:differentFrom	*AddDifferentFrom* *DeleteDifferentFrom*	*Individual*	*Individual*
owl:disjointWith	*AddDisjointWith* *DeleteDisjointWith*	*Class*	*Class*
owl:equivalentClass	*AddEquivalentClass* *DeleteEquivalentClass*	*Class*	*Class*
owl:equivalentProperty	*AddEquivalentProperty* *DeleteEquivalentProperty*	*ObjectProperty* */DatatypeProperty*	*ObjectProperty* */DatatypeProperty*
owl:FunctionalProperty	*AddFunctionalProperty* *DeleteFunctionalProperty*	*ObjectProperty* */DatatypeProperty*	
owl:hasValue	*AddHasValue* *DeleteHasValue*	*ObjectProperty* */DatatypeProperty*	*Individual* */DataValue*
owl:Individual	*AddIndividual* *DeleteIndividual*	*Individual*	
owl:intersectionOf			
owl:InverseFunctionalProperty	*AddInverseFunctionalProperty* *DeleteInverseFunctionalProperty*	*ObjectProperty*	
owl:inverseOf	*AddInverseOf* *DeleteInverseOf*	*ObjectProperty*	*ObjectProperty*

[22] *XML Schema Datatype : http://www.w3.org/TR/xmlschema-2/*

owl:maxCardinality	AddMaxCardinalityProperty DeleteMaxCardinalityProperty	ObjectProperty	xsd:integer
owl:minCardinality	AddMinCardinalityProperty DeleteMinCardinalityProperty	ObjectProperty	xsd:integer
owl:Nothing			
owl:ObjectProperty	AddObjectProperty DeleteObjectProperty	ObjectProperty	
owl:oneOf			
owl:sameAs	AddSameAs DeleteSameAs	Individual	Individual
owl:someValuesFrom	AddSomeValuesFrom Delete SomeValuesFrom	ObjectProperty /DatatypeProperty	Individual /DataValue
owl:SymmetricProperty	AddSymmetricProperty DeleteSymmetricProperty	ObjectProperty	
owl:Thing			
owl:TransitiveProperty	AddTransitiveProperty Delete TransitiveProperty	ObjectProperty	
owl:unionOf			
rdfs:domain	AddDomainProperty DeleteDomainProperty	ObjectProperty /DatatypeProperty	Class
rdfs:range	AddRangeProperty DeleteRangeProperty	ObjectProperty /DatatypeProperty	Class /xsd :datatype
rdfs:subClassOf	AddSubClassOf DeleteSubClassOf	Class	Class
rdfs:subPropertyOf	AddSubObjectPropertyOf DeleteSubObjectPropertyOf AddSubDatatypePropertyOf DeleteSubDatatypePropertyOf	ObjectProperty /DatatypeProperty	ObjectProperty /DatatypeProperty

Table 11. OWL DL Basic Change Operator Types Displayed by OntoVersioning According to OWL DL Ontological Constructs

According to Table 11, a list of 70 change operator types is directly derived from the set of OWL ontological constructs. Note that the owl:unionOf and owl:intersectionOf OWL DL constructs have not yet been tailored as change operator types. Also, to reach a $\mathcal{SHOIN}(\mathcal{D})$ equivalent set of change operators, the datatype, object property and datatype property instantiations operator types should also be added to the list. These improvements are planned for future developments.

According to the list of OWL DL change operator types, changes can be instantiated with IRIs of the ontology elements typed with the corresponding types. Before modelling any change, domain experts have to first load the project as coded below.

```
Project pp = new Project("ontologies", "test");
```

Secondly, they have to load the VersionGraph associated to the project ontology as coded below.

```
VersionGraph vg = pp.loadLastVersion();
```

Third, they can instantiate the changes and add them to the VersionGraph factory. As described in previous part, each change corresponds to an instantiation of a subclass of *Operation*. Therefore, changes have to be secondly coded as such, in order to assign the correct parameters types, and then recreate the corresponding links between the *Operation* instantiations in the future ontology VersionGraph version. Coding a change, roughly consists in building an *Operation* instance and apply it on the required ontology elements. Once the whole changes are coded they have to be added to the current VersionGraph factory. For instance the addition of a change adding a class Person is coded as followed:

```
vg.getOperations().addClass("http://example.org/#Person");
```

Then the library makes the necessary checks to see whether the change can be applied as such, namely by a structural and logical consistency preventive assessment. This verification is automatically done, whatever the verification mode, before implementing the changes. It verifies the whole set of changes at once. If no inconsistency is found, changes are implemented on a copy of the ontology. Otherwise, if the verification detects problems, such as structural or logical inconsistencies, changes are not applied. In case of logical inconsistencies, if the chosen verification mode is the one powered by OntoVersioning, i.e. CLOCk, the domain experts are noticed of the minimal inconsistent set of changes identified in the list of changes modelled.

2.3 Implementing and Validating Changes

This part shows how to implement, save the changes and how to generate of a new version of the ontology.

When changes are assessed consistent and if the domain experts consider the ontology has reached a satisfying state, they can launch their implementation on a copy of the ontology. A copy of the ontology VersionGraph is then updated by adding the consistent change operations represented in the VersionGraph factory. Metadata related to the changes can also be added to the VersionGraph. These new changes are serialized and linked with those contained by the VersionGraph. The VersionGraph serialization launching the change implementation is coded as follows:

```
vg.serializeVersionGraph();
```

Once the VersionGraph is serialized, the domain experts can call a second logical consistency check using the Pellet Reasoner. First, an empty ontology model of the considered evolved ontology copy using the Pellet SPEC has to be created as follows.

```
String ont = "http://example.org/#";
OntModel model =
      ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC);
 model.read( ont );
```

Second the logical consistency of the evolved ontology copy can be assessed by checking the model with Pellet as follows.

```
boolean isConsistent =
      ((PelletInfGraph)model.getGraph()).getKB().isConsistent();
```

If inconsistencies are detected, domain experts can ask to visualize the corresponding report. In the other case, they can validate the implementation to propagate the changes to the potential sub-

ontologies defined by views on the current ontology, or to the source ontology, if the ontology is actually a sub-ontology defined by a views on another ontology. The propagation phase process is described in the fourth section.

After this phase, the domain experts can commit the corresponding ontology version. The following instruction creates the new ontology version from the last serialized VersionGraph :

```
OwlConvector owlc =
      new OwlConvector(vg,"ontologies","test",vg.getMaxVersion());
```

However, if after the implementation phase, the domain experts are not satisfied by the evolved ontology result, they can proceed to another evolution process taking as input this evolved copy. Also, if they prefer a previous evolution, they can commit the corresponding ontology version from a previous version of the VersionGraph. For instance, the instruction below creates the ontology from the second version of the ontology VersionGraph.

```
OwlConvector owlc = new OwlConvector(vg, "ontologies", "test", 2);
```

2.4 Discussion

This part consists in a discussion on the limits of the change management features of the OntoVersioning API according to tests results and comparison with existing solutions.

Testing the validity of the entire change management features of the API, cannot be exhaustively realized as it would require the development of many OWL DL ontologies, with different way of specifying them, e.g. with or without anonymous nodes, with more or less inconsistencies etc. Tests have however been realized on case studies for the evolution process, the structural and logical consistency preventive approaches and the versioning process.

Concerning the evolution process including the structural consistency approach, the limits lie in the absence of certain OWL DL basic change operator types identified previously and of usual composed changes. Two displayed change operator types causes compilation errors when instantiated. This issue will be resolved in future developments. However, the application of changes, corresponding to the use of the other displayed ones, shows great results: the ontology and its corresponding VersionGraph have always been structurally consistent updated.

Concerning the logical consistency preventive approach, the CLOCk technique, which, as said previously, is not complete at all (contains only unsatisfiability constraints of 9 change operator types), has been tested on a toy ontology inconsistency identification example, quite similar to the one described in the OntoVersionGraph methodology formalization (c.f. Chapter 5). The expected inconsistencies were found and the corresponding minimal inconsistent set of changes was returned in few seconds. The result is quite promising for the future implementation of the full OWL DL set of change operator types. However no scalability tests have been realized with the model checking approach.

Concerning the versioning process, every ontology and VersionGraph versions, committed after ontology evolution or rollback to a previous version, were successfully generated.

In a nutshell, the change management module should go through a phase of functional testing on a real application case to validate the algorithms, so as to improve its functioning.

Comparing the OntoVersioning API change management with an existing implemented solution gives a better idea of the limits of its functional scope. Among the collaborative OWL change management solutions most recently implemented, the ProtegeHGDB Plugin features can be compared to the

OntoVersioning change management features. ProtegeHGDB[23] is a Protégé plugin, which provides interesting features for collaborative local version management of OWL ontologies. It allows storing ontologies in a fully transactional hypergraph database[24]. The displayed features are:

- A live database storage (with no need to save) with export/import to/from all OWL formats.
- Version management with full history, commit, rollback, revert, etc. operations.
- Team features including ontology sharing on a common repository and managing versions following the Subversion model with a centralized location. The versioning is done at the ontology level and all operations work with axioms and change sets on axioms.

The ProtegeHGDB functional scope is definitely dedicated to ontology versioning features. The versioning process follows three steps. First, the system detects ontology modifications realized by different collaborators with the Protégé editor. Second, it stores the corresponding axioms in a database, allowing the collaborators to realize several modifications on the ontology before committing a new version. Third the committed version is exported as an OWL ontology in which are also represented the whole set of modifications since the previous committed version.

This plugin shows several contributions regarding ontology versioning features, which are complementary to those proposed by OntoVersioning. Compared to the OntoVersioning API, the main ones remain in displaying a collaborative ontology edition environment with a live database dedicated to change storage and a common ontology repository. These are the main features that cannot be implemented in the actual API but, which are required to allow domain experts to collaboratively work with the API. Among the other versioning features, only one should not be retained for the API improvements: the representation of the ontology modifications within its terminology, which is one of the alternative solutions to logs of changes for change representation. As explained in Chapter 3, it is not adapted for formal ontologies, which terminology should only describes the knowledge of a certain domain and should not contain any other kind of information. Also in OntoVersioning, the use of logs is not only dedicated to versioning features of formal ontologies, but for many other change management purposes, such as change consistency checking and resolution, view specification and sub-ontology extraction, and change impact management. The ProtegeHGDB Plugin remains an interesting solution to complete the functional scope of the Ontology Change Management features.

[23] *ProtegeHGDB Plugin : http://sharegov.org/#!../protegehgdb/owltools.html*
[24] *Hypergraph database : http://www.hypergraphdb.org/index*

3 Specifying Views & Extracting Sub-Ontologies in OntoVersioning

This section details the use of the view specification and sub-ontology extraction module in the OntoVersioning API. A first part explains how to define a view according to key elements of a sub-domain to describe. A second part explains how to extract the sub-ontology according to the view definition from the source ontology and how to generate the corresponding sub-ontology VersionGraph. A third part presents the results of the tests realized on this module.

View extraction is articulated in two steps. First the representation of the view definition. Second its application as a mapping on the ontology. The ontology can be an ontology created by OntoVersioning or an external OWL DL ontology.

3.1 Defining a View

This part explains how to define a view according to key elements of a sub-domain to describe.

If domain experts want to extract a sub-ontology describing a sub-domain of an ontology related domain, they first need to chose its corresponding key elements among the classes, object properties, attribute properties and individuals described in the ontology. These key elements, called starters are the input parameters of a view definition which results is a sub-ontology. A view definition in OntoVersioning is supported by a mapping, which applied on the ontology will allow to extract their axiomatic entourage, i.e. subsets of the ontology set of axioms. Defining the mapping amounts, for the domain experts, to add the different starter elements' IRIs to the mapping list. For instance, the instructions below define a view from the IRIs of three starter elements.

```
ArrayList<String> mapping = new ArrayList<>();
mapping.add( "http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream" );
mapping.add( "http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza" );
mapping.add( "http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping"
);
```

The system verifies the existence of each IRI within the ontology. If they all exist, then the extraction can be done. Otherwise the missing IRIs are noticed to the domain experts and the extraction process stops. Instructions below respectively specify the ontology to check, then verify it according to the view definition and notices the user of the result.

```
String ontoUri = "pizza.owl";

ModuleVerifMapping m = new ModuleVerifMapping(ontoUri, mapping);

if (! m. Check())
      m.displayMappingErrors();

else
      System.out.println("URIs exist, Sub-Ontology Extraction can be
done");
```

3.2 View Extraction

This part explains how to extract the sub-ontology defined by a mapping.

When the mapping is checked, domain experts can choose to extract the corresponding sub-ontology and to save it in an OWL DL file. Then need to indicate the name of the view target file before

launching the extraction. For instance, the instruction below specifies the file, which will contains the sub-ontology resulting from the extraction.

```
String viewIri = "view1.owl";

The following instruction launches the sub-ontology extraction module in
order to specify the ontology and the view definition.

ModuleExtractSubOntology e =
      new ModuleExtractSubOntology(ontoUri,mapping, viewUri);
```

The extraction algorithm follows a set of structural and logical rules inspired by the complete and partial extraction rules defined in the OntoVersionGraph methodology (c.f. Chapter 6). They ensure the consistent extraction of the axioms composing the axiomatic entourage of each starter element. The algorithm is handled by the translation of the extraction rules into the corresponding SPARQL queries, allowing to select each axioms and automatically join the whole sets in a OWL DL sub-ontology. However SPARQL queries can manipulate resources identified by an IRI not anonymous ones. Anonymous nodes, i.e. anonymous resources, are then not supported in this API. Unfortunately, anonymous nodes are very often used to describe classes in existing ontologies. Resolving this issue is part of the development perspectives envisioned in the future. The last instructions below respectively run the extraction of the sub-ontology and save the result in the previous OWL DL file.

```
e.extract();
e.save();
```

When the sub-ontology is extracted, domain experts can choose to serialize its VersionGraph in order to trace it in the versioning system. Instructions below create the VersionGraph corresponding to the sub-ontology from the source ontology VersionGraph and attach it to the sub-ontology OWL DL file.

```
ModuleExtractSubVersionGraph m = new ModuleExtractSubVersionGraph(vg,
mapping, viewIri);

    The source ontology VersionGraph is automatically updated by the
addition of the corresponding change representing the view specification.
```

3.3 Discussion

This part consists in a discussion on the limits of the View Specification and Sub-Ontology Extraction features of the OntoVersioning API according to tests results.

Like for the change management process, functional tests have been realized on the view specification and sub-ontology extraction process implemented in OntoVersioning, but there are not enough exhaustive to demonstrate the validity of the implementation. For instance, a view has been defined on the Travel OWL DL ontology[25] proposed by the Protégé editor tool as example ontology. The mapping was parameterized with the classes *Capital, City* and *LuxuryHotel*. The sub-ontology extraction provided a satisfying view result as illustrated in Figure 38. Indeed the whole axiomatic entourage of each class has been extracted as expected.

[25] *Travel ontology : http://protege.cim3.net/file/pub/ontologies/travel/travel.owl*

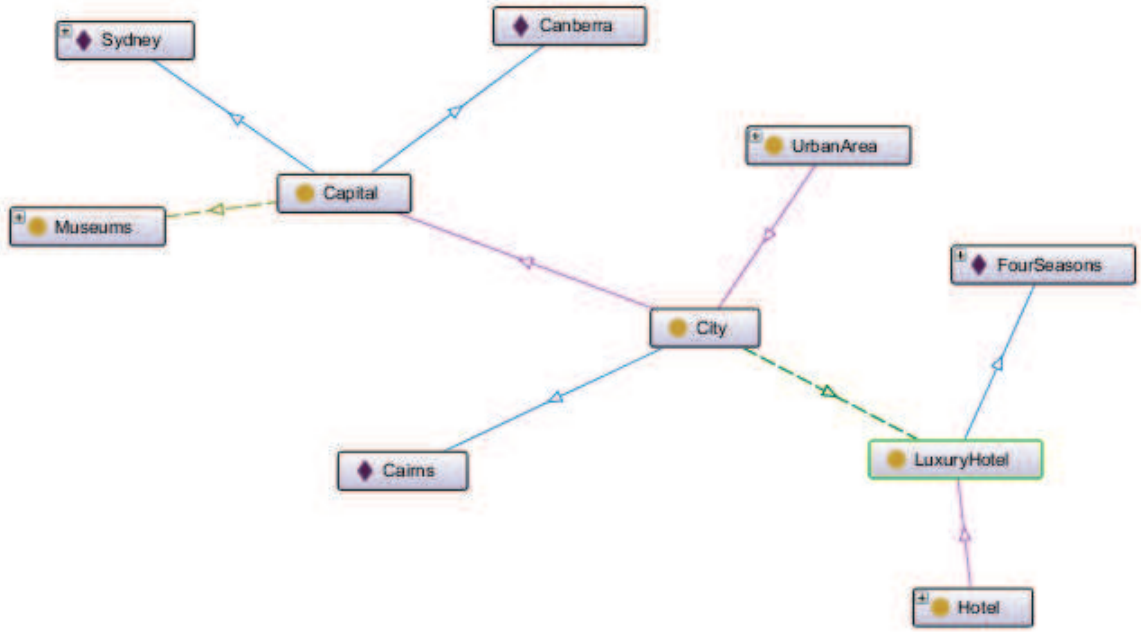


Figure 38. View Result of a Sub-Ontology Extraction from Travel Ontology

This result is promising as the ontology structural and logical consistency was kept. However, further tests have to be done to give a more accurate estimation of the process quality.

Concerning the execution performance, finding an accessible equivalent system with which comparing the sub-ontology extraction was impossible. Therefore, quantities of tests have been made in order to quantify the runtime, according to two parameters: the size of the ontology and the size of sub-ontologies extracted from the ontology. However, if there are many ontologies available online, few have the essential character to be imported within OntoVersioning: their compliance with OWL DL logical constraints allowing the Pellet reasoner to check them as consistent. This criterion greatly reduced the choice, but some tests could, however, be made on the remaining ontologies. However, this solution being not quite complete, as no control is permitted over the ontologies content. Some consisted only of a terminology, others were too simple, or, conversely, too complex and required to load several gigabytes. To measure the scalability of the sub-ontology extraction module provided by OntoVersioning, a system to automatically generate ontologies was required, in order to avoid the time consuming task of modelling a formal OWL DL ontology, which in this case is not really pertinent. Two important parameters have been retained for the generation of these ontologies : the total number of classes in the ontology , as well as the minimum number of links between classes. These parameters set the size and the complexity of the ontology.

The protocol for automatic generation of ontology is relatively, and voluntarily simple, and can be described as follows. First, the desired number of random classes is created. Second, for each class, we randomly choose a class to bind to another, and also randomly selects a link between the two classes (e.g. equivalence or disjunction constraints, etc.). This class is marked to avoid indefinitely reusing the same class for the next linking. The operation is repeated as many times as necessary in order to reach the minimum number of links between each class. The final ontology obtained is then serialized. Indeed, ontologies generated using this protocol are not true OWL DL ontologies, as they only contain classes as ontological elements. However, using them to test the starter classes' extractions is enough to assess the extraction approach feasibility. Many random ontologies have been generated following this protocol,

with at least two links between each class and containing respectively 1, 10, 100, 1000, 10000, and 50000 classes. From the whole set of automatically generated ontologies, it is then possible to pass the tests in order to quantify the system performance. However some measure constraints have to be respected:

- Measuring only the execution time of the sub-ontology extraction process
- Doing the same tests *n* times, to obtain an average measure. Indeed, more than on any other platform, the performance of the same program can widely vary depending on the state of the JVM. More the JVM works, more it is efficient and faster it can process.
- Not disturbing the measurements performed by running other programs. So one should not use the test machine during program execution. Starting a minimal system would also be a good idea.

A series of increasing large mappings are applied for each ontology. This allows to see how the system reacts with scalability. As a reminder, prior to extraction, the system analyses all classes specified in the mapping, and classes for the explicit definition of the latter. This is where the ontology automated generation protocol shows a contribution. It allows to measure the extraction performance between ontologies that are composed of the same number of classes, but with a different number of minimum links between them. The following table and graph shows the extraction results obtained from the extraction of sub-ontologies containing respectively 1, 10, 100, 1000 , 10000 and 50000 classes from the a set of ontologies generated with at least five links between their classes and containing respectively 1, 10, 100, 1000 , 10000 and 50000 classes.

Sub-Ontology Extraction Time (ms)		**Number of Classes to Extract**					
		1	**10**	**100**	**1000**	**10000**	**50000**
Number of Classes Contained in Ontology	**1**	13					
	10	26	61				
	100	61	111	351			
	1000	194	145	343	1701		
	10000	1357	1321	2336	3046	11282	
	50000	7259	7718	7368	7875	15569	49408

Table 12. Sub-Ontology Extraction Time w.r.t. Number of Classes to Extract & Number of Classes Contained in Ontology

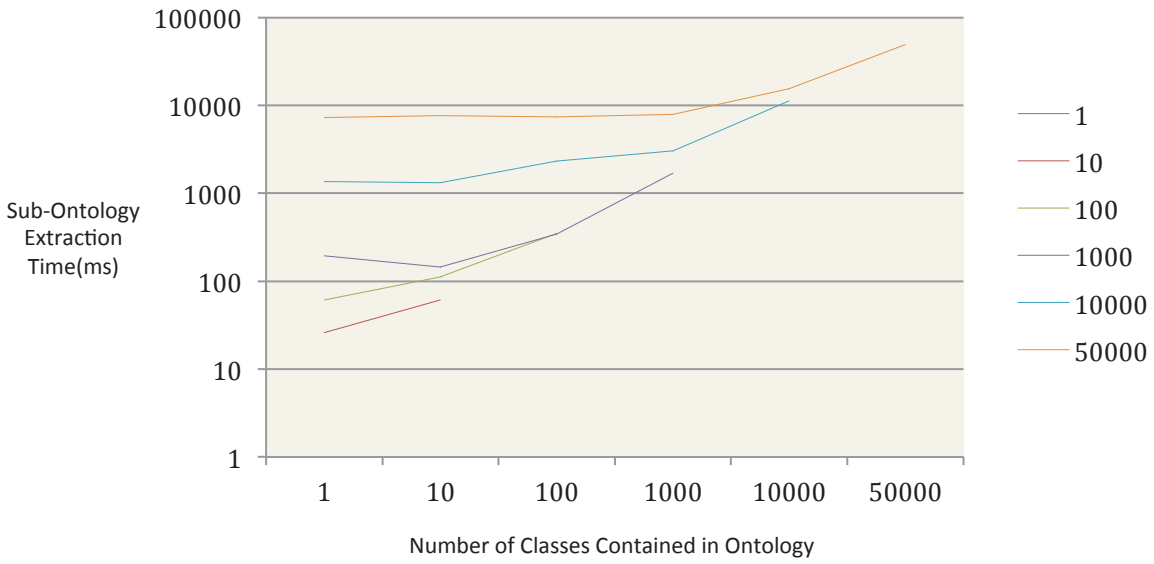


Figure 39. Sub-Ontology Extraction Time w.r.t. Number of Classes to Extract & Number of Classes Contained in Ontology

According to Table 12 and Figure 39, the sub-ontology extraction time is exponentially growing w.r.t. the number of classes contained in the ontology and the number of classes to extract. However the extraction time measurements are highly satisfying: less than 7 seconds for the extraction of a sub-ontology composed of 1000 classes from an ontology containing 50000 classes with at least 5 links between each class.

4 Using Change Impact Management Module in OntoVersioning

This section details the use of the change impact management module within the OntoVersioning API. The two propagation approaches formalized in the OntoVersionGraph methodology, respectively Top-Down and Bottom-Up (c.f. Chapter 7), are implemented in the API. Domain experts can use them during the change propagation phase of an ontology evolution, after the serialization of the ontology VersionGraph following the change implementation and before committing a new ontology version. The use of the Top-Down and or the Bottom-Up propagation approaches are respectively presented in the first and in the second part of this section. A third part presents the results of the tests realized on this module.

4.1 Using the Top-Down Propagation Approach

This part explains how to use the Top-Down propagation approach to manage the impact of changes applied on an ontology when propagated to its views.

The propagation of changes from the evolved copy of an ontology to its views is divided into two steps. The impact assessment and the corresponding update of each view definition is the first task to achieve. This task, if necessary, occurs during the propagation phase of the project. The second is the re-extraction of each sub-ontology defined by views on the evolved copy of the ontology, from the application of each view definitions on the evolved ontology even if they have not been updated. Domain experts first need to load the VersionGraph of the changes applied during the last ontology evolution as follows.

```
VersionGraph vg = pp.loadLastVersion();
VersionGraph subvg=vg.getLastOperations() ;
```

Then they just need to build an instance of the *ModuleTopDown* with this VersionGraph and the IRI of a view specified on this ontology as parameters, to launch the Top Down Propagation module.

```
ModuleTopDown f = new ModuleTopDown(subvg,viewUri);
```

They can then launch the update of the corresponding view mapping as follows:

```
f.update();
```

The update algorithm automatically creates the list of parameters of each deletion change contained in the VersionGraph, which are similar to starters elements of the view mapping to assess the impact of the ontology evolution on the view definition. If the list is not null, it then deletes all the starter elements from the mapping according to the list processed. The sub-ontology re-extraction process is similar to its first extraction. Domain expert just need to set the corresponding view definition, even if not updated, the IRI of the ontology evolved copy, as parameters of a new instance of *ModuleExtractSubOntology*. It implies coding the following instructions:

```
ModuleExtractSubOntology e = new ModuleExtractSubOntology(ontoUri,
mapping, viewUri );
e.extract();
e.save();
```

When the sub-ontology is extracted, domain experts can choose to serialize its VersionGraph in as follows.

```
ModuleExtractSubVersionGraph m = new ModuleExtractSubVersionGraph(vg,
mapping, viewIri);
```

4.2 Using the Bottom-Up Propagation Approach

This part explains the use of the Bottom-Up propagation approach to propagate the changes applied on a OWL DL sub-ontology defined by a view on a OWL DL source ontology to this ontology and its other views. The propagation of changes from a sub-ontology to the ontology from which it was extracted and, by extension, to its other views, is divided into three steps.

The sub-ontology evolution constitutes the first step. The evolution process is the same as a classic OWL DL ontology one. Instructions below load the last VersionGraph of the sub-ontology and apply the changes required by the view sub-domain experts.

```
Project p2 = new Project("ontologies", "view");
VersionGraph vg2 = p2.loadLastVersion();
vg2.getOperations().addObjectProperty("http://example.org/#friendOf");
vg2.getOperations().addClass("http://example.org/#friend");
vg2.getOperations().addDomainProperty("http://example.org/#friendOf","http:
//example.org/#friend");
vg2.getOperations().addRangeProperty("http://example.org/#friendOf","http:/
/example.org/#friend");
```

Like for any OWL DL ontology, consistency is resolved by the structural and logical preventive checks according to the chosen verification mode. If changes are assumed consistent, the sub-domain experts then need to implement the changes on the sub-ontology by serializing a copy of its VersionGraph with the changes, so that they can check again its logical consistency with Pellet.

```
vg2.serializeVersionGraph();

String subOnt = "http://example.org/view.owl";
OntModel model2 =
      ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC);
model2.read( subOnt );
boolean isConsistent =
      ((PelletInfGraph)model2.getGraph()).getKB().isConsistent();
```

When changes are implemented and checked, changes can be propagated to the ontology, calling a second evolution process. The second step consists then in applying the changes implemented on the sub-ontology on the source ontology. Sub-domain experts have to build an instance of the ModuleBottomUp Class setting the serialize VersionGraph copy of the evolved sub-ontology and the source ontology IRI as parameters, to launch the Bottom-Up Propagation module.

```
ModuleBottomUp g = new ModuleBottomUp(vg2,ontoIri);
```

They can then launch the update of the corresponding ontology as follows:

```
g.update();
```

The update algorithm automatically extracts the corresponding changes from the sub-ontology serialized VersionGraph copy, add them to the ontology VersionGraph, implement them and check their consistency on the ontology. If changes are assessed consistent on the whole ontology, then they can be

propagated to the other views defined on it. The third step consists then in the propagation of the changes applied on the ontology to the other views. Domain experts have to extract the last changes applied on the evolved ontology in a VersionGraph and set it as parameter of a new instance of the *ModuleTopDown* class with the different sub-ontologies' IRI. Then they need to call the corresponding update method to update the view definition. Then they need to launch the corresponding sub-ontology extraction module and VersionGraph extraction module to extract the new sub-ontology version from the evolved copy of the ontology and save its new VersionGraph version. The piece of code below illustrates its use for one view.

```
VersionGraph subvg=vg.getLastOperations() ;
ModuleTopDown f = new ModuleTopDown(subvg,viewIri1);
f.update();
ModuleExtractSubOntology e
      = new ModuleExtractSubOntology(ontoIri,mapping,viewIri1 );
e.extract();
e.save();
ModuleExtractSubVersionGraph m = new ModuleExtractSubVersionGraph(vg,
mapping, viewIri);
```

This process must be repeated for each sub-ontology defined by a view on the ontology. After these sub-ontologies are all updated, the ontology evolution can be validated by generating the corresponding ontology version as below.

```
OwlConvector owlc =
      new OwlConvector(vg,"ontologies","test",vg.getMaxVersion());
```

4.3 Discussion

The change impact management module implemented in the OntoVersioning is the only feature, which has not been really tested yet. Only few unit tests have been made to validate the consistent update feature of a view mapping impacted by changes propagated from an ontology. The rest of this module features being based on the change management module and the sub-ontology extraction module, it was considered enough to show the feasibility of the Top-Down and Bottom-Up Propagation approaches. A series of functional tests are however envisioned to allow improving the corresponding features. Also, no equivalent solution was found to allow a functional scope or an execution performance comparison.

5 Conclusion and Future Development Perspectives

This chapter has described the implementation of the OntoVersionGraph ontology change management methodology.

It has first given a general overview on the API dedicated to this approach, called OntoVersioning. The three main modules corresponding to the three main blocks of the methodology are: the change management module, embedding the evolution and versioning processes using structural and logical preventive resolution modules, the view specification and sub-ontology extraction module, and the change impact management module. These modules are all supported by VersionGraph files containing the global logs of changes of ontologies or sub-ontologies. A VersionGraph being represented as an OWL light ontology describing the whole changes applied on an ontology or a sub-ontology and their related metadata, can be queried by SPARQL queries for many purposes, such as logical inconsistency identification, ontology version retrieving and regeneration, etc. The development and the evolution of an OWL DL ontology is until now realised from scratch but the API is also envisioned to support existing OWL DL ontologies upon the automated construction of its VersionGraph.

The use of each of the OntoVersioning API modules has been secondly explained with code examples, in order to show how they can be integrated in an existing system and precisely describe their functional scope. The three main modules do not embed all the features described by the methodology, but a major part of it. They have not been tested on a concrete project. However, the change management module features have been assessed on some cases of study, as well as the view specification and sub-ontology extraction module, which has also been tested for scalability. The change impact management module is the only feature, which have not been tested yet. A series of functional tests are however envisioned to allow improving the corresponding features. Globally the results obtain after manually using each of these modules were satisfying : OWL DL ontologies used as input have been consistently evolved, versioned and views have been specified on them, the corresponding sub-ontologies consistently extracted and managed with the consistent propagation of changes between them. Also the related VersionGraphs were generated and updated as expected.

For the moment, the limits identified in the three main modules concern the technical and the functional levels. Concerning the change management module, technical issues are met with the absence of certain OWL DL change operator types and with the incomplete set of change unsatisfiabilities displayed by the logical consistency preventive module. The functional issues are mostly related to the absence of a collaborative environment in which integrating the API. Concerning the view specification and sub-ontology extraction module, technical issues are met when dealing with anonymous nodes. The functional issues are related to the absence of the datatype and datavalue starter types defined in the methodology as extraction rules. Concerning the change impact management module, the only functional issue identified is the absence of the collaborative environment to allow automatic notifications to the sub-domain experts, which views are impacted by change propagation.

Considering the whole API, functional issues are met with the use of the Jena ontology management API. In Jena, ontologies in use are loaded in memory, which can cause a scalability problem if they are too complex. Therefore, some large ontologies exceeding a certain size cannot be handled by the OntoVersioning API, as they are not supported by Jena. A mean to resolve this issue would be to associate an OWL triple store, in order to avoid loading in memory large ontologies and limits the access to the ontology by queries. Finally, the API can be complex to use by non-computer scientists, as it does not propose a GUI to display a user-friendly menu for the change management process, or to visualize the evolution of an ontology, the view extraction or the propagation of changes.

References

Gueffaz, M., Rampacek, S., & Nicolle, C. (2011, August). RDF2NμSMV: mapping semantic graphs to NμSMV model checker. In AFIN 2011, The Third International Conference on Advances in Future Internet (pp. 49-53).

Gueffaz, M., Pittet, P., Rampacek, S., Cruz, C., & Nicolle, C. (2012). Inconsistency Identification In Dynamic Ontologies Based On Model Checking. INSTICC, ACM SIGMIS, 418-421.

Chapter 9

Conclusion & Future Works

Summary

This chapter constitutes the conclusion of my thesis. The first section sums up the contributions of my thesis proposal. The second section gives a list of perspectives of research and future works.

Plan

1 Contribution 229
2 Research Perspectives 231

The **world changes over time**, impacting the knowledge of every subdomain it contains. Therefore, systems describing the knowledge of a certain domain should be able to consider changes occurred to keep its knowledge representation up-to-date. **Formal ontologies** are one of them: they explicitly and formally represent the knowledge of a domain in all its forms and modes of existence (Cochiarella, 1991). Collaboratively developed, a formal ontology **allows the domain users to understand each other** by sharing the **same terminology despite the different assumptions** they have on the domain conceptualization (Guarino, 1995). However, due to its completeness, the **complexity of its conceptualization** can sometimes make the domain knowledge inaccessible for its users. Also domain users often need to access only a subset of this knowledge. Light sub-portions of the ontology, i.e. views, defined by their assumptions should then be produced to provide domain users a **personalized and comprehensive view of the domain knowledge** described by the ontology. Considering such ontology, the world changes may impact both domain knowledge and user's assumptions and their application can turn inconsistent the knowledge represented by the ontology and its views. An **inconsistent ontology is no longer usable** by the domain community. Therefore, the ontology must be consistently evolved so that its users can continue to cooperate and understand each other. For my thesis, the problem was the following: **how to enable collaborative ontology change management considering changes in the domain and new domain user's assumptions**? Until now, existing research in the Semantic Web did not allow such a change management for formal ontologies. Therefore, the contribution of my thesis is the **design of an approach to ontology change management dedicated to formal ontologies** enabling the three following features, **the specification of ontology views based on new domain user's assumptions, a consistent collaborative evolution of the ontology and its views, and the change impact management between them**. This last chapter describes these contributions in a first section and gives a list of research perspectives and future works in a second section.

1 Contribution

This section describes the contribution of my thesis in a summary. In order to resolve the problem described previously, and design an ontology change management methodology dedicated to formal ontologies enabling the three features identified, this thesis has developed an approach through eight chapters.

Chapter 2 has addressed the **specific context of the thesis**, Formal Ontology, through the point of view of the Applied Ontology orientation. It has demonstrated the importance of ontologies for solving the semantic heterogeneity of information systems. It has showed that my subject is more specifically related to the problem of **heterogeneity of conceptualization** adressed by **formal ontologies**. Then, it has addressed the characteristics of formal ontologies, its terminological composition, its interpretation namely in a knowledge base, its inference services which allow inferring new knowledge from the terminological and assertionnal levels, its quality assessment according to its precision and correctness and the formal nature of its representation language. It has concluded that DLs or formal languages derived from them, such as the broadly used OWL DL (equivalent to $\mathcal{SHOIN}(\mathcal{D})$ DL) and the recent OWL 2, are the Semantic Web best suitable languages for the representation of a formal ontology. This is why this thesis specifically **focuses on the DL $\mathcal{SHOIN}(\mathcal{D})$** ontology language. According to these specificities, it has deduced the proposal of my thesis should produce a **formal methodology for managing evolution and versioning of a formal ontology and its user's assumptions** on the domain. It has identified three objectives, **(1) enable change management of formal ontologies** since their creation, **(2) enable adaptation of formal ontologies to new user assumptions** and **(3) the coherent change impact propagation** between a formal ontology and its different assumptions. In order to ensure the integrity of a formal ontology during its lifecycle, it has proposed the following eight criteria:

1. Proposing a formal **ontology evolution methodology** to enable conceptual modelling subsequent to an ontological analysis;
2. Specifying **ontology and changes in $\mathcal{SHOIN}(\mathcal{D})$** or equivalent formal language;
3. Managing formal ontologies **representing all the user assumptions** on its domain;
4. Managing the **logical consistency**, i.e. related to axioms, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages;
5. Management of **structural consistency**, i.e. related to the representation language, in $\mathcal{SHOIN}(\mathcal{D})$ or equivalent formal languages with formal representation changes;
6. **Evolution and Versioning** of a formal ontology and its user assumptions since its creation;
7. **Impact analysis and change propagation** applied to an assumption to all the other assumptions described by the ontology;
8. **Adaptation** of the ontology to a new user's assumption.

Most of these issues are being addressed in the **Ontology Change Management** (OCM) and **Ontological Views** research areas. These criteria have been the guidelines of the two states of art presented in Chapter 3. Many of the key solutions among the existing ones have been identified in these research areas to match the eight criteria. Only the 7th criterion still remains unsatisfied in existing works. Considering this 7th criterion, **three remaining deadlocks** have been identified for my thesis proposal.

1. **Extraction of $\mathcal{SHOIN}(\mathcal{D})$ ontology views** modelled as an ontological change, traceable in the ontology log of changes;
2. The **creation of a log of changes** for each ontology view to allow it evolving and being versioned like the original $\mathcal{SHOIN}(\mathcal{D})$ ontology;
3. The **impact management** by the propagation of the changes applied from the ontology to its views and from a view to the ontology and the other views, based on the log of changes of each of them.

To cope with these three deadlocks and the three objectives stated before, a **proposal articulated in three blocks** has been proposed.

1. The **first block** consists in the definition of a **collaborative change management process ensuring consistent representation and applications of changes on a $\mathcal{SHOIN}(\mathcal{D})$ ontology**. An ontology model suited for $\mathcal{SHOIN}(\mathcal{D})$ is formalized to model structurally consistent changes by guarantying the expressivity level of the ontology after their application. An additional structural consistency preventive approach based on axiom dependencies resolution is performed at the end of the change modelling phase. A preventive logical consistency qualification of these changes is included at the change semantics phase in order to help the user ensure their logically consistent application. A log of changes is generated and updated after each validated phase in order to display the versioning features of the methodology such as change propagation. This first block has been described in Chapter 4 and Chapter 5. Chapter 4 has formalized the $\mathcal{SHOIN}(\mathcal{D})$ ontology model and its related basic and composed changes modelling. Chapter 5 has formalized the preventive structural and logical consistency approaches.
2. The **second block** consists in the **logical specification and extraction of ontology views as $\mathcal{SHOIN}(\mathcal{D})$ sub-ontologies**. The definition of a view has to be traceable like a change in the ontology global log of changes. A global log of changes has to be generated for each view extracted in order to display the versioning features of the methodology on the views. This second block has been described in Chapter 6.
3. The **third block** consists in the **impact management with the consistent propagation of changes** applied from the ontology to its views (Top-Down approach) and, inversely from a view to the ontology and the other views (Bottom-Up Approach). This third block has been formalized in Chapter 7.

Chapter 8 has presented an implementation of the proposal for my thesis in the form of Java API that supporting OWL DL ontologies, called OntoVersioning. It has given a general overview on the API features. But it has also focused on the **three main modules** corresponding to the three main blocks of the methodology, which are the **change management module**, embedding the evolution and versioning processes using structural and logical preventive resolution modules, the **view specification and sub-ontology extraction module**, and the **change impact management module**.

To summarize, the major contribution of this approach lies in the **formalization of a formal ontology change management methodology** (Pittet & al, 2010, Pittet & al, 2012) extended to formal ontology views management (Pittet & al, 2014). This extension lies, firstly, in the **conception of a view extraction process producing true sub-ontologies** from an ontology. It, secondly, stands in a **change propagation process handling changes impact between the views and the ontology** from which they are derived. The side contributions are: (1) design of an ontological model dedicated to the modeling of changes for $\mathcal{SHOIN}(\mathcal{D})$ ontologies (Pittet et al, 2013.), (2) design of a preventive approach to maintain the structural consistency based on syntactic constraints of $\mathcal{SHOIN}(\mathcal{D})$ (Pittet et al., 2013a), and (3) the design of a preventive approach to help domain experts resolve logical consistency, based on the identification of potential inconsistent axioms sets that each new application of changes could cause (Gueffaz & al, 2012).

2 Research Perspectives

This section gives a non-exhaustive list of the **limitations of the proposal** presented in my thesis and introduces some **research perspectives**.

The main limitations of my proposal lie in its **formalization,** its **functional coverage** and its **implementation.**

First, the methodology is **only applicable on ontologies represented in the $\mathcal{SHOIN}(\mathcal{D})$ description logic** or any formal language with an equivalent expressiveness (such as OWL). It can, nevertheless, be easily extended to any other formal language by adapting the list of changes to this language list of constructors.

Second, from a functional point of view, the method **does not handle the change detection phase**, which allows domain experts to understand the need for change of the ontology as a whole. This phase is nevertheless included in the methodology and could be supplemented for example with a semi-automatic ontology learning approach. Also, when propagating changes from one view to the global ontology and from the global ontology to the other views, changes are automatically detected as they are propagated through a log of changes.

Third, the methodology **does not propose a method of corrective resolution of logical consistency** dedicated to formal ontologies. During the change implementation phase of the ontology evolution process, a corrective logical consistency check of the application of changes on the ontology is carried out using the Pellet reasoner. However, Pellet, although sound, is not totally complete for the $\mathcal{SHOIN}(\mathcal{D})$ DL. Also specific resolution strategies for inconsistencies identified by both Pellet approach and the logical consistency preventive approach are not provided.

Fourth, at the implementation level, the prototype is a Java API extending the Jena ontology management API. It **only supports RDF, RDF-S and OWL ontologies including OWL DL**. It **does not propose a GUI** to visualize the evolution of an ontology nor the view extraction or the propagation of

changes. The ontology and its views are loaded in memory during the change management process, which can cause a **scalability problem if the ontology is too complex**. Also, the logical consistency preventive module of the prototype, CLOCk, is not complete. In future works, it might be replaced by new implementation substituting the model checking technique by the use of SPARQL 1.1 queries, suited for the use of regular expressions, which can help identifying patterns of minimal inconsistent changes in ontology logs as they are represented by OWL instances. It may also improve the scalability of the module to handle large ontologies consistency maintenance. Concerning the view extraction, it is already carried out by SPARQL queries, but it causes different problems in this context. As this language is dedicated to querying RDF graphs, **the extraction of OWL DL axioms** and not only RDF triples shows some tricky issues, namely the one related to anonymous nodes. A query language dedicated to formal ontologies would be more appropriate. Finally, the **API does not support the entire methodology**, namely the collaborative environment required to apply it, and some technical issues still need to be resolved.

I currently work on these improvement perspectives to resolve the different limitations identified in the methodology and the prototype. As additional research perspective, the methodology can easily be **extended to support $\mathcal{SROIQ}(\mathcal{D})$ OCM,** improving the expressivity level of the ontologies supported and making possible the application of the methodology on OWL 2 ontology profiles. Finally **the integration of the prototype in a collaborative environment** of a real application domain, would allow practically experimenting the whole methodology and more efficiently assessing its contribution.

References

Cocchiarella, N. B. (1991). Formal ontology.

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. International journal of human-computer studies, 43(5), 625-640.

Guarino, N., & Musen, M. A. (2005). Applied ontology: Focusing on content. Applied Ontology, 1(1), 1-5.

Gueffaz, M., Pittet, P., Rampacek, S., Cruz, C., & Nicolle, C. (2012). Inconsistency Identification In Dynamic Ontologies Based On Model Checking. INSTICC, ACM SIGMIS, 418-421.

Pittet, P., Cruz, C., & Nicolle, C. (2010). Towards Dynamic Ontology-Integrating Tools of Evolution and Versionning in Ontology. Towards Dynamic Ontology-Integrating Tools of Evolution and Versionning of Ontology.

Pittet, P., Nicolle, C., & Cruz, C. (2012). Guidelines for a dynamic ontology-integrating tools of evolution and versioning in ontology. arXiv preprint arXiv:1208.1750.

Pittet, P., Cruz, C., & Nicolle, C. (2013, January). A Structural SHOIN (D) Ontology Model for Change Modelling. In On the Move to Meaningful Internet Systems: OTM 2013 Workshops (pp. 442-446). Springer Berlin Heidelberg.

Pittet, P., Cruz, C., & Nicolle, C. (2013a, September). Modeling Changes for SHOIN (D) Ontologies: An Exhaustive Structural Model. In Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on (pp. 104-109). IEEE.

Pittet, P., Cruz, C., Nicolle, C. (2014). "Ontology Views for Ontology Change Management: A state of the art". In IGI Global Encyclopedia of Information Science and Technology (3rd Ed.).

List of Figures

Figure 1. Semantic Web Stack....21
Figure 2 – One Conceptualisation for Multiple Representations....37
Figure 3 – Different Conceptualizations of the same Representation....37
Figure 4 – Ontology Definitions Classifications according to their type of Heterogeneity Resolution....38
Figure 5 – Reference Triangle (on the left) and a Possible Representation (on the right)....41
Figure 6 – Formal Ontology Building Process According to Applied Ontology....41
Figure 7 – Description Logic Formal Ontology Interpretation Domain....44
Figure 8. Unsatisfiable Ontology Model....45
Equation 1. Inference Problem Reduction to Subsumption Problems....45
Equation 2. Inference Problem Reduction to Satisfiability Problem....45
Figure 9. Inconsistent Ontology Model....46
Figure 10– Result of a Formal Ontology Development Process....47
Figure 11 – Expressivity of a 12 Symbol Language....49
Figure 12 – Expressivity of a 18 Symbol Language....49
Figure 13: Change Types Classification According to the Three Causes of Changes of an Ontology....62
Figure 14: OntoVersionGraph Evolution Process....92
Figure 15. Graphical Representation of the Ontology $\mathcal{S}$O with the G-MOT Editor.....103
Figure 16. $\mathcal{SHOIND}$ Ontology Simplified Interpretation Domain....122
Figure 17. Interpretation of an Equivalence Constraint on Two Concepts D1 and D2....123
Figure 18. Hierarchy of Extraction rules defined for the Extraction of a $\mathcal{SHOIN}(\mathcal{D})$ View....153
Figure. 19. Graphical Representation of the Ontology Example $\mathcal{S}$**O** with the G-MOT Editor 161
Figure 20. Starter Concept *Person* Extraction Result....163
Figure 21. Starter Role *teaches* Extraction Result....164
Figure 22. Starter Instance *christophe1* Extraction Result....165
Figure 23: Sub-Ontology $\mathcal{S}$**O**′ Extraction Result....167
Figure 24. Top-Down Propagation Approach Steps....178
Figure 25. Bottom-Up Approach....182
Figure 26. The Architect Point of View on a Wall (on the left) & the Electrician Point of View on the same Wall (on the right)....185
Figure 27. Inconsistent Brute Application of the Modification from the Electrician Point of View....186
Figure 28. The Wall as a Grid of Wall Spaces with the Inconsistent Brute Application....186
Figure 29. Consistent Application of the Modification from the Electrician Point of View....186
Figure 30. The Wall as a Grid of Wall Spaces with A Consistent Application....187
Figure 31. Inconsistent Application of the Modification from the Electrician Point of View w.r.t. the New Model Constraint....190
Figure 32. The Ground as a Grid of Ground Spaces with An Inconsistent Application....191
Figure 33. Consistent Application of the Modification from the Electrician Point of View with the New Model Constraint....191
Figure 34. The Wall as a Grid of Wall Spaces with A Consistent Application with the New Model Constraint....191
Figure 35. OntoVersioning Simplified Versioning Process....201
Figure 36. OntoVersioning Extended Versioning Process....201
Figure 37. Concept Operation Object Model....206
Figure 38. View Result of a Sub-Ontology Extraction from Travel Ontology....217
Figure 39. Sub-Ontology Extraction Time w.r.t. Number of Classes to Extract & Number of Classes Contained in Ontology....219

List of Tables

Table 1 – Primitives, Interpretation and Main Characteristics of the Ontological Level according to the Other Study Levels....40
Table 2 – Usual Semantic Web Representation Languages....50
Table 3. Complexity Classes According to Description Logics Expressivity Level....51
Table 4. Comparison of OCM proposals w.r.t. my Thesis 8 criteria....72
Table 5. The list of $\mathcal{SHOIN}(\mathcal{D})$ Basic Change Operator Types and their Parameters....106
Table 6. Structural Dependencies Between Ontological Elements and Axiom Subsets....118
Table 7. Unsatisfiability Constraints of the AddEquivalentConcept Change Operator Type Formalized as Generic Inconsistent Axioms Sets....124
Table 8a. $\mathcal{SHOIND}$ Composed Change Example Unsatisfiability Constraints part 1....125
Table 8b. $\mathcal{SHOIN}(\mathcal{D})$ Composed Change Example Unsatisfiability Constraints part 2....126
Table 8c. Composed Change Example Unsatisfiability Constraints part 3....126
Table 9. Logical Consistency Preventive Analysis Process Results Example....139
Table 10. Complete and Partial Extractions Distribution according to Structural Constraints and Extraction Rule Types....152
Table 11. OWL DL Basic Change Operator Types Displayed by OntoVersioning According to OWL DL Ontological Constructs....211
Table 12. Sub-Ontology Extraction Time w.r.t. Number of Classes to Extract & Number of Classes Contained in Ontology....218

Annex: $\mathcal{SHOIN}(\mathcal{D})$ Basic Addition Changes Unsatisfiability Constraints

AddDisjointConcept

The AddDisjointConcept change corresponds to the addition of a disjointness constraint between two concepts. If two concepts D1 and D2 are disjoint, their intersection is unsatisfiable. These change unsatisfiability constraints are also applied on D1 and D2 equivalent concepts, sub-concepts and intersected concepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Concepts D1, D2	$\delta_C(\text{D1})_i$=(D1,D2)	4. if $\leq_C (D1) = \{(\text{D1}, \text{D1}_1), \dots, (\text{D1}, \text{D1}_n)\}$ and ∃i such that $\leq_C (D1) = (D1, \text{D2}_i)$
		5. if $\leq_C (D2) = \{(\text{D2}, \text{D2}_1), \dots, (\text{D2}, \text{D2}_n)\}$ and ∃i such that $\leq_C (D2) = (D2, \text{D1}_i)$
		6. if ∃i $\varepsilon_C(\text{D1})_i$=(D1,D2)
		7. if ∃i $\varepsilon_C(\text{D2})_i$=(D2,D1)
		8. if ∃i $\sqcap_C(\text{D1})_i$=(D1,D2)
		9. if ∃i $\sqcap_C(\text{D2})_i$=(D2,D1)
		0. if ∃i $-_C(\text{D2})_i$=(D2,D2-) and ∃j $\delta_C(\text{D1})_j$=(D1, D2-)
		1. if ∃i $-_C(\text{D1})_i$=(D1,D1-) and ∃j $\delta_C(\text{D2})_j$=(D2, D1-)
		2. if $\iota_C(D2)$={(D2,I2_1),…,(D2,I2_n)} and ∃i such that $\iota_C(D1)_i$= (D1,I2_i)
		3. if $\iota_C(D1)$={(D1,I1_1),…,(D1,I1_n)} and ∃j such that $\iota_C(D2)_j$= (D2,I1_i)
		4. if ∃i, j $\iota_C(D2)$={(D2,I2_1),…,(D2,I2_n)} and $\iota_C(D1)$={(D1,I1_1),…,(D1,I1_n)} such that $\varepsilon_I(\text{I1}_i) = (\text{I1}_i, \text{I2}_j)$
		5. if ∃i, j $\iota_C(D2)$={(D2,I2_1),…,(D2,I2_n)} and $\iota_C(D1)$={(D1,I1_1),…,(D1,I1_n)} such that $\varepsilon_I(\text{I2}_j) = (\text{I2}_i, \text{I1}_i)$

AddSubConcept

The AddSubConcept change corresponds to the addition of an inclusion constraint between two concepts. If a concept D2 is a subconcept of a concept D1, then the intersection of D2 with the complement of D1 is unsatisfiable. These change unsatisfiability constraints are also applied on D1 and D2 equivalent concepts and D2 subconcepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Concepts D1, D2	$\leq_C(\text{D1})_i$=(D1,D2)	• If ∃i $-_C(\text{D1})_i$=(D1,D1-) and $\leq_C(\text{D1-})$={(D1-,D1-_1),…,(D1-,D1-_n)} and ∃j ∃k such that $\varepsilon_C(\text{D2})_k$=D2, D1-_j)
		• if ∃i $-_C(\text{D1})_i$=(D1,D1-) and ∃j $\varepsilon_C(\text{D2})_j$=(D2, D1-)
		• if ∃i $\delta_C(\text{D1})_i$=(D1,D2)
		• if ∃i $\delta_C(\text{D2})_i$=(D2, D1)
		• if ∃i $-_C(\text{D1})_i$=(D1,D1-) and ∃j $\leq_C(\text{D1-})_j$=(D1-, D2)
		• If ∃i $-_C(\text{D1})_i$=(D1,D1-) and $\iota_C(D2)$={(D2,I2_1),…,(D2,I2_n)} and ∃j $\iota_C(\text{D1-})_j$=(D1-, I2_j)
		1. If ∃i $-_C(\text{D1})_i$=(D1,D1-) and $\iota_C(D2)$={(D2,I2_1),…,(D2,I2_n)} and $\iota_C(\text{D1-})$={(D1-,I1-_1),…,(D1-,I1-_n)} and ∃j ∃k such that $\varepsilon_I(\text{I1-}_j)_j = (\text{I1-}_j, \text{I2}_k)$

AddComplementConcept

The AddComplementConcept change corresponds to the addition of a complementarity constraint between two concepts. If a concept D2 is a complement of a concept D1, then the intersection of D2 with the complement of D1 is unsatisfiable. These change unsatisfiability constraints are also applied on D1 and D2 equivalent concepts and subconcepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints

Concepts D1, D2	$-_C(D1)_i=(D1,D2)$	2. If $\exists i \leq_C (D2)_i = (D2, D1)$ 3. If $\exists i \leq_C (D1)_i = (D1, D2)$ 4. if $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$and $\exists i$ such that $\leq_C (D2)_i = (D2, D1_i)$ 5. if $\leq_C (D2) = \{(D2, D2_1), \dots, (D2, D2_n)\}$and $\exists i$ such that $\leq_C (D1)_i = (D1, D2_i)$ 6. if $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$ and $\leq_C (D2) = \{(D2, D2_1), \dots, (D2, D2_n)\}$ and $\exists i$ $\exists j$ such that $\varepsilon_C(D2_j) = (D2_j, D1_i)$ 7. if $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$ and $\leq_C (D2) = \{(D2, D2_1), \dots, (D2, D2_n)\}$ and $\exists i$ $\exists j$ such that $\varepsilon_C(D1_i) = (D1_i, D2_j)$ 8. if $\iota_C(D1)$={(D1,$I1_1$),…,(D1,$I1_n$)} and $\exists$i such that $\iota_C(D2)_i$= (D2,$I1_i$) 9. if $\iota_C(D2)$={(D2,$I2_1$),…,(D2,$I2_n$)} and $\exists$i such that $\iota_C(D1)_i$= (D1,$I2_i$) 10. if $\iota_C(D1)$={(D1,$I1_1$),…,(D1,$I1_n$)} and $\iota_C(D2)$={(D2,$I2_1$),…,(D2,$I2_n$)} and $\exists$i $\exists$j $\exists$k such that $\varepsilon_I(I1_j)_k$= ($I1_j$, $I2_i$) 11. if $\iota_C(D1)$={(D1,$I1_1$),…,(D1,$I1_n$)} and $\iota_C(D2)$={(D2,$I2_1$),…,(D2,$I2_n$)} and $\exists$i $\exists$j $\exists$k such that $\varepsilon_I(I2_j)_k = (I2_j, I1_i)$ 12. If $\exists i$ $\varepsilon_C(D2)_i$=(D2, D1) 13. If $\exists i$ $\varepsilon_C(D1)_i$=(D1,D2) 14. If $\exists i$ $-_C(D1)_i$=(D1,D1-) and $\exists j$ $\delta_C(D1\text{-})$=(D1-, D2) 15. If $\exists i$ $-_C(D2)_i$=(D2,D2-) and $\exists j$ $\delta_C(D2\text{-})$=(D2-, D1) 16. If $\exists i$ $-_C(D1)_i$=(D1,D1-) and $\exists j$ $\delta_C(D2)$=(D2, D1-) 17. If $\exists i$ $-_C(D2)_i$=(D2,D2-) and $\exists j$ $\delta_C(D1)$=(D1, D2-)

AddConceptIntersection

The AddConceptIntersection change corresponds to the definition of an intersection of several concepts Di. These change unsatisfiability constraints are also applied on D1 and D2 equivalent concepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Concepts D1…Dn	$\sqcap_C(D)$=(D1,…,Dn)	• If $\exists i \leq_C (Di) = \{(Di, Di_1), \dots, (Di, Di_n)\}$ and If $\exists j$ $-_C(Di_j)_j$=(Di_j,Dj_j) • If $\exists i \leq_C (Di) = \{(Di, Di_1), \dots, (Di, Di_n)\}$and $\exists j \exists k \exists l$ such that $\delta_C(Di_j)_j$=(Di_j,Dk_l) • If $\exists i$ $\exists j$ $-_C(Di)_i$=(Di,Dj) — If $\exists i$ $-_C(Di)_i$=(Di,Di-) and $\exists j$ $\iota_C(Dj)$={(Dj,Ij_1),…,(Dj,Ij_n)} and $\exists k$ $\iota_C(Di\text{-})_k$= (Di-,Ij_k) — If $\exists i$ $-_C(Di)_i$=(Di,Di-) and $\exists j$ $\exists k$ $\iota_C(Dj)_k$=(Dj,Ij_k) and $\exists l$ $\iota_C(Di\text{-})_l$=(Di-,$I\text{-}_l$) such that $\varepsilon_I(Ij_k) = (Ij_k, I\text{-}_l)$ — If $\exists i$ $-_C(Di)_i$=(Di,Di-) and $\exists j$ $\exists k$ $\iota_C(Dj)_k$=(Dj,Ij_k) and $\exists l$ $\iota_C(Di\text{-})_l$=(Di-,$I\text{-}_l$) such that $\varepsilon_I(I\text{-}_l) = (I\text{-}_l, Ij_k)$ — If $\exists i$ $-_C(Di)_i$=(Di,Di-) and $\exists j$ $\varepsilon_C(Dj)$=(Dj, Di-) — If $\exists i$ $\exists j$ $\delta_C(Di)$=(Di, Dj)

AddSubDatatype

The AddSubDatatype change corresponds to the addition of inclusion constraints between two datatypes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Datatypes T1, T2	$\leq_T(T1)_i$=(T1,T2)	• If $\exists i \leq_T (T2)_i = (T2, T1)$

AddEquivalentRole

The AddEquivalentRole change corresponds to the addition of an equivalence constraint between two roles R1 and R2. If two roles R1 and R2 are equivalent, the intersection of R1 and the complement of R2, and the intersection of R2 with the complement of R1 are unsatisfiable. These change unsatisfiability constraints are also applied on R1 and R2 equivalent roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints

Roles R1, R2	$\varepsilon_R(R1)_i$=(R1,R2)	1. If ∃i $-_R(R1)_i$=(R1,R1-) and $\leq_C (R2) = \{(R2, R2_1), \dots, (R2, R2_n)\}$and ∃j, ∃k such that $\leq_R$(R-)$_j$=(R-, $R2_k$) • If ∃i $-_R(R2)_i$=(R2,R2-) and $\leq_R (R1) = \{(R1, R1_1), \dots, (R1, R1_n)\}$ such that ∃j, ∃l $\leq_R$(R2-)$_j$={ $R1_l$} • If ∃i $-_R(R2)_i$=(R2,R2-) and ∃j $\varepsilon_R(R1)_j$=(R1, R2-) • If ∃i $-_R(R1)_i$=(R1,R1-) and ∃j $\varepsilon_R(R2)$j=($R2$, R1-) • If ∃i $\kappa_R(R1)_i$=(R1,KR_1) and ∃j $\kappa_R(R2)_j$= (R2,KR_2) such that $KR_1 \neq KR_2$ • If ∃i $-_R(R1)_i$=(R1,R2) • If ∃i $-_R(R2)_i$=(R2,R1) • If ∃i $-_R(R1)_i$=R_{-1} and ∃j $-_R(R2)_j$=R_{-2} and $-_R(R_{-1}) = (R_{-1}, R_{-2})$ • If ∃i $-_R(R1)_i$=R_{-1} and ∃j $-_R(R2)_j$=R_{-2} and $-_R(R_{-2}) = (R_{-2}, R_{-1})$ • maxCardR(R1)=n1 and maxCardR(R2)=n2 such that n1≠ n2 • minCardR(R1)=n1 and minCardR(R2)=n2 such that n1≠ n2 • $\rho_{\exists R}(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\rho_{\exists R}(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that (R,($D1_i$,$Ra1_i$)) ≠ (R,($D2_j$,$Ra2_j$)) • $\rho_{\forall R}(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\rho_{\forall R}(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that (R,($D1_i$,$Ra1_i$)) ≠ (R,($D2_j$,$Ra2_j$)) • $\rho_R(R1)$={(R,($D1_1$,i$Ra1_1$)),…,(R,($D1_n$,i$Ra1_1$))} and $\rho_R(R2)$={(R,($D2_1$,i$Ra2_1$)),…,(R,($D2_n$,i$Ra2_1$))} and ∀i, ∃j such that (R,($D1_i$,i$Ra1_i$)) ≠ (R,($D2_j$,i$Ra2_j$)) • $\sigma_R(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\sigma R(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that (R,($D1_i$,$Ra1_i$)) ≠ (R,($D2_j$,$Ra2_j$)) • if $\iota_R(R1)$={(R,(i$D1_1$,i$Ra1_1$)),…,(R,(i$D1_n$,i$Ra1_1$))} and $\iota_R(R2)$={(R,(i$D2_1$,i$Ra2_1$)),…,(R,(i$D2_n$,i$Ra2_1$))} and ∀i, ∃j such that (R1,(i$D1_i$,i$Ra1_i$)) ≠,(R2,(i$D2_j$,i$Ra2_j$))

AddInverseRole

The AddInverseRole change corresponds to the addition of an inverse constraint between two roles R1 and R2. If two roles R1 and R2 are inverses than the intersection of R1 and R2 is unsatisfiable. These change unsatisfiability constraints also apply on R1 and R2 equivalent roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ **Model Axiom**	**Terminological Inconsistent Axioms**
Roles R1, R2	$-_R(R1)$=(R1,R2)	• if $\leq_R (R2) = \{(R2, R2_1), \dots, (R2, R2_n)\}$ and ∃i, j such that $\leq_{Rsub} (R1)$={ $R2_j$} • if $\leq_R (R1) = \{(R1, R1_1), \dots, (R1, R1_n)\}$ and ∃i, j such that $\leq_{Rsub} (R2)$={ $R1_i$} • If ∃i $\varepsilon_R(R1)_i$=(R1, R2) • If ∃i $\varepsilon_R(R2)_i$=($R2$, R1) • If ∃i $\kappa_R(R1)_i$=(R1,KR_1) and ∃j $\kappa_R(R2)_j$=(R2,KR_2) such that KR_1=KR_2 and $KR_1 \neq \emptyset$ and KR_1≠ Transitive and KR_1≠ Symmetric • If ∃i $\kappa_R(R1)_i$=(R1,KR_1) and ∃j $\kappa_R(R2)_j$=(R2,KR_2) such that $KR_1 \neq KR_2$ and [$KR_1 = \emptyset$ or KR_1=Transitive or KR_1=Symmetric or KR_2=Transitive or KR_2=Symmetric or (KR_1=Functional and KR_2≠InverseFunctional) or (KR_1=InverseFunctional and KR_2≠Functional)] • If ∃i $-_R(R2)_i$=(R2,R2-) and $-_R(R1)$=(R1,R2-) • If ∃i $-_R(R1)_i$=(R1,R1-) and $-_R(R2)$=(R2,R1-) • maxCardR(R1)=n1 and minCardR(R2)=n2 such that n1≠ n2 • minCardR(R1)=n1 and maxCardR(R2)=n2 such that n1≠ n2 • $\rho_{\exists R}(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\rho_{\exists R}(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that $D1_i$≠ $Ra2_j$ or $Ra1_i \neq D2_j$, • $\rho_{\forall R}(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\rho_{\forall R}(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that $D1_i$≠ $Ra2_j$ or $Ra1_i \neq D2_j$, • $\rho_R(R1)$={(R,($D1_1$,i$Ra1_1$)),…,(R,($D1_n$,I$Ra1_1$))} and $\rho_R(R2)$={(R,($D2_1$,i$Ra2_1$)),…,(R,($D2_n$,I$Ra2_1$))} and ∀i, ∃j such that $D1_i$≠ i$Ra2_j$ or $IRa1_i \neq D2_j$ • $\sigma_R(R1)$={(R,($D1_1$,$Ra1_1$)),…,(R,($D1_n$,$Ra1_1$))} and $\sigma R(R2)$={(R,($D2_1$,$Ra2_1$)),…,(R,($D2_n$,$Ra2_1$))} and ∀i, ∃j such that $D1_i$≠ $Ra2_j$ or $Ra1_i \neq D2_j$ • if $\iota_R(R1)$={(R,(I$D1_1$,I$Ra1_1$)),…,(R,(I$D1_n$,I$Ra1_1$))} and $\iota_R(R2)$={(R,(I$D2_1$,I$Ra2_1$)),…,(R,(I$D2_n$,I$Ra2_1$))} and ∀i, ∃j such that I$D1_i$≠ I$Ra2_j$ or $IRa1_i \neq ID2_j$

AddSubRole

The AddSubRole change corresponds to the addition of an inclusion constraint between two roles R1 and R2. If a role R2 is a subrole of a role R1, the intersection of R2 and the inverse of R1 are unsatisfiable. These change unsatisfiability constraints are also applied on R1 and R2 equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Roles R1, R2	$\leq_R(R1)_i=(R1,R2)$	• If $\exists i\ -_R(R1)_i=(R1,R1\text{-})$ and $\varepsilon_R(R2)=(R2, R1\text{-})$ • If $\forall i\ \exists j\ \kappa_R(R1)_i=(R1, KR_1)$ and $\kappa_R(R2)_j=(R2, KR_2)$ such that $KR_1 \neq KR_2$ • If $\exists i\ -_R(R1)_i=(R1,R2)$ • If $\exists i\ -_R(R2)_i=(R2,R1)$ • If maxCardR(R1)=n1 and maxCardR(R2)=n2 such that n1≠ n2 • If minCardR(R1)=n1 and minCardR(R2)=n2 such that n1≠ n2 • If $\rho_{\exists R}(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\rho_{\exists R}(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(D1_i)_j=(D1_i,D1_i\text{-})$ and $\exists k\ \exists l\ \leq_C(D2_k)_l=(D2_k, D1_i\text{-})$ • If $\rho_{\exists R}(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\rho_{\exists R}(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(Ra_i)_j=(Ra_i,Ra_i\text{-})$ and $\exists k\ \exists l\ \leq_C(Ra_k)_l=(Ra_k, Ra_i\text{-})$ • If $\rho_{\forall R}(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\rho_{\forall R}(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(D1_i)_j=(D1_i,D1_i\text{-})$ and $\exists k\ \exists l\ \leq_C(D2_k)_l=(D2_k, D1_i\text{-})$ • If $\rho_{\forall R}(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\rho_{\forall R}(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(Ra_i)_j=(Ra_i,Ra_i\text{-})$ and $\exists k\ \exists l\ \leq_C(Ra_k)_l=(Ra_k, Ra_i\text{-})$ • If $\rho_R(R1)=\{(R1,(D1_1,IRa1_1)),\ldots,(R1,(D1_n,IRa1_1))\}$ and $\rho_R(R2)=\{(R2,(D2_1,IRa2_1)),\ldots,(R2,(D2_n,IRa2_1))\}$ and $\exists i\ \exists j$ such that $-_C(D1_i)_j=(D1_i,D1_i\text{-})$ and $\exists k\ \exists l\ \leq_C(D2_k)_l=(D2_k, D1_i\text{-})$ • If $\rho_R(R1)=\{(R1,(D1_1,IRa1_1)),\ldots,(R1,(D1_n,IRa1_1))\}$ and $\rho_R(R2)=\{(R2,(D2_1,IRa2_1)),\ldots,(R2,(D2_n,IRa2_1))\}$ and $\forall i, \exists j$ such that $iRa2_j \neq iRa1_i$ • If $\sigma_R(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\sigma_R(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(D1_i)_j=(D1_i,D1_i\text{-})$ and $\exists k\ \exists l\ \leq_C(D2_k)_l=(D2_k, D1_i\text{-})$ • If $\sigma_R(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\sigma_R(R2)=\{(R2,(D2_1,Ra2_1)),\ldots,(R2,(D2_n,Ra2_1))\}$ and $\exists i\ \exists j$ such that $-_C(Ra_i)_j=(Ra_i,Ra_i\text{-})$ and $\exists k\ \exists l\ \leq_C(Ra_k)_l=(Ra_k, Ra_i\text{-})$ • if $\iota_R(R2)=\{(R2,(ID2_1,IRa2_1)),\ldots,(R2,(ID2_n,IRa2_n))\}$ and $\sigma_R(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\exists i$ such that $-_C(D1)_i=(D1,D1\text{-})$ and $\exists j$ such that $\iota_C(D1\text{-})=(D1\text{-},ID2_j)$ • if $\iota_R(R2)=\{(R2,(ID2_1,IRa2_1)),\ldots,(R2,(ID2_n,IRa2_n))\}$ and $\sigma_R(R1)=\{(R1,(D1_1,Ra1_1)),\ldots,(R1,(D1_n,Ra1_1))\}$ and $\exists i$ such that $-_C(Ra1)_i=(D1,Ra1\text{-})$ and $\exists j$ such that $\iota_C(Ra1\text{-})=(Ra1\text{-},IRa2_j)$ • if $\exists i\ -_R(R1)_i=(R1, R1\text{-})$ and $\exists j \leq_R (R2)_j=(R2, R2_j)$ such that $\exists k\ \leq_R(R2_j)_k=(R2_j, R1\text{-})$

AddRoleSignature

The AddRoleSignature change corresponds to the addition of a domain and range concept type constraint to a role R. These change unsatisfiability constraints are also applied on R equivalent roles and subroles, on D and Ra equivalent concepts and subconcepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints

Role R Concepts D, Ra	σ_R =(R,(D,Ra))	(R)$_i$

2. If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$and $\exists i \leq_R (R)_i$=(R, R$_i$) and $\exists j\ \sigma_R$(R$_i$)$_k$=(R$_i$(D$_i$,Ra$_i$)) such that $\exists k\ -_C$(D$_i$)$_k$=(D$_i$,D)
3. If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$and $\exists i \leq_R (R)_i$=(R, R$_i$) and $\exists j\ \sigma_R$(R$_i$)$_k$=(R$_i$(D$_i$,Ra$_i$)) and$\exists k\ -_C$(D)$_k$=(D,D-) such that $\exists l \leq_C$(D-)$_l$=(D-, D$_i$)
4. If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and $\exists i \leq_R (R)_i$=(R, R$_i$) and $\exists j\ \sigma_R$ (R$_i$)$_k$=(R$_i$(D$_i$,Ra$_i$)) such that $\exists k\ -_C$(Ra$_i$)$_k$=(Ra$_i$,Ra)
5. If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$and $\exists i \leq_R (R)_i$=(R, R$_i$) and $\exists j\ \sigma_R$(R$_i$)$_k$=(R$_i$(D$_i$,Ra$_i$)) and$\exists k\ -_C$(Ra)$_k$=(Ra,Ra-) such that $\exists l \leq_C$(Ra-)$_l$=(Ra-, Ra$_i$)
6. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(DR-,RaR-)) and $-_C$(RaR-)$_j$=(RaR-,RaR--) and $\exists k\ \leq_C$(D)$_k$=(D, RaR--)
7. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(DR-,RaR-)) and $\exists k\ \delta_C$(D)$_k$=(D, RaR-)
8. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(DR-,RaR-)) and $\exists k\ \delta_C$(RaR-)$_k$=(RaR-,D)
9. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(D-R,Ra-)) and $-_C$(DR-)$_j$=(DR-,DR--) and $\exists k\ \leq_C$(Ra)$_k$=(Ra, DR--)
10. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(DR-,Ra-)) and $\exists k\ \delta_C$(DR-)$_k$=(DR-, Ra)
11. If $\exists i\ -_R$(R)$_i$=(R,R-) and $\exists j\ \sigma_R$(R-)=(R-,(DR-,RaR-)) and $\exists k\ \delta_C$(Ra)$_k$=(Ra,DR-)
12. If $\iota_R(R)$={(R,(ID$_1$,IRa$_1$)),...,(R,(ID$_n$,IRa$_1$))} and $\exists i\ -_C$(D)$_i$=(D,D-) and $\exists j$ such that ι_C(D-)$_j$=(D-,ID$_j$)
13. If $\iota_R(R)$={(R,(ID$_1$,IRa$_1$)),...,(R,(ID$_n$,IRa$_1$))} and and $\exists i\ -_C$(Ra)$_i$=(Ra,Ra-) and $\exists j$ such that ι_C(Ra-)=(Ra-,IRa$_j$)
14. If $\iota_R(R)$={(R,(ID$_1$,IRa$_1$)),...,(R,(ID$_n$,IRa$_1$))} and $\exists i\ -_C$(D)$_i$=(D,D-) and $\exists j\ \leq_C$(D-)$_j$=(D-, D$_j$-)and $\exists k\ \exists l$ such that ι_C(D$_j$-)$_k$=(D$_j$-,ID$_l$)
15. If $\iota_R(R)$={(R,(ID$_1$,IRa$_1$)),...,(R,(ID$_n$,IRa$_1$))} and and $\exists i\ -_C$(Ra)$_i$=(Ra,Ra-) and $\exists j \leq_C$(Ra-)$_j$=(Ra-, Ra$_j$-) and $\exists k\ \exists l$ such that ι_C(Ra$_k$-)=(Ra$_k$-,IRa$_l$)

AddRoleFunctionalProperty

The AddRoleFunctionalProperty change corresponds to the addition of a functional property to a role R. These change unsatisfiability constraints are also applied on R equivalent roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R	κ_R(R)=(R,Functional)	• if $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and $\exists i\ \forall j$ such that κ_R(R$_i$)$_j \neq$ (R$_i$,Functional) • If $\exists i$ such that ε_R(R)$_i$=(R,Re$_i$) and $\forall j\ \kappa_R$(Re$_i$)$_j \neq$ (Re,Functional) • If $\exists i$ such that $-_R$(R)$_i$=(R,R-$_i$) and$\forall j\ \kappa_R$(R-$_i$)$_j \neq$(R-, InverseFunctional) • if $\iota_R(R)$={(R,(iD$_1$,iRa$_1$)),...,(R,(iD$_n$,iRa$_1$))} and $\exists i, \exists j$ such that iD$_i$= iD$_j$ and iRa$_i \neq$iRa$_j$ • if $\iota_R(R)$={(R,(iD$_1$,iRa$_1$)),...,(R,(iD$_n$,iRa$_1$))} and $\exists i, \exists j$ such that iD$_i \neq$ iD$_j$ and iRa$_i$=iRa$_j$

AddRoleTransitiveProperty

The AddRoleTransitiveProperty change corresponds to the addition of a transitive property to a role R. These change unsatisfiability constraints are also applied on R equivalent roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R	κ_R(R)$_i$=(R,Transitive)	• If $\exists i$ such that ε_R(R)$_i$=(R,Re$_i$) and $\forall j$ such that κ_R(Re$_i$) $\neq$ (Re$_i$,Transitive) • if $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and $\exists i\ \forall j$ such that κ_R(R$_i$)$_j \neq$ (R$_i$,Transitive)

AddRoleSymmetricProperty

The AddRoleSymmetricProperty change corresponds to the addition of a transitive property to a role R. These change unsatisfiability constraints are also applied on R equivalent roles, inverse roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R	κ_R(R)$_i$=(R,Symmetric)	• if $\leq_R (R) = \{(D1, iRa1), \dots, (R, R_n)\}$ and $\exists i \forall j$ such that κ_R(R$_i$)$_j \neq$ (R,Symmetric) • If $\exists i$ such that ε_R(R)$_i$=(R,Re$_i$) and $\forall j\ \kappa_R$(Re$_i$)$_j \neq$ (R,Symmetric) • If σ_R(R)= {(R,(D$_1$,Ra$_1$)),...,(R,(D$_n$,Ra$_1$)) and $\exists i$ such that D$_i \not\equiv$ Ra$_i$ • if $\iota_R(R)$={(R,(iD$_1$,iRa$_1$)),...,(R,(iD$_n$,iRa$_1$))} and $\exists i$ such that iD$_i \in$D$_i$ and iRa$_i \notin$ Ra$_i$

AddRoleInverseFunctionalProperty

The AddRoleInverseFunctionalProperty change corresponds to the addition of an inverse functional property to a role R. These change unsatisfiability constraints are also applied on R equivalent roles and subroles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints

Role R	$\kappa_R(R)_i$=(R,InverseFunctional)	• if $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and ∃i ∀j such that $\kappa_R(R_i)_j \neq$(R,Functional) • If ∃i ∀j such that $\varepsilon_R(R)_i$=(R,Re) and $\kappa_R(Re)_j \neq$ (R,InverseFunctional) • If ∃i, ∀j such that $-_R(R)_i$=(R,R_i-) and $\kappa_R(R_i\text{-})_j \neq$(R,Functional)

AddRoleUniversalRestriction

The AddRoleUniversalRestriction change corresponds to the addition of an universal restriction to a role R. These change unsatisfiability constraints are also applied on R equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R Concept C	$\rho_{\forall R}(R)_i$=(R,D)	• If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and ∃i $\rho_{\forall R}(R_i) = \{(R_i, D_1), \dots, (R_i, D_n)\}$ such that ∀j $D_i \not\equiv D$ • If ∃i $\varepsilon_R(R)_i$=(R,Re) and $\rho_{\forall R}(Re) = \{Re, De_1), \dots, (Re, De_n)\}$ such that ∀j $D \neq De_j$ • if $\iota_R(R)$={(R,(iD_1,iRa_1)),…,(R,(iD_n,iRa_1))} and ∃i such that ∀j $iRa_i \notin$ D

AddRoleExistentialRestriction

The AddRoleExistentialRestriction change corresponds to the addition of an existential restriction to a role R. These change unsatisfiability constraints are also applied on R equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R Concept D	$\rho_{\exists R}(R)_i$=(R,D)	• If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and ∃i $\rho_{\exists R}(R_i) = \{(R_i, D_1), \dots, (R_i, D_n)\}$ such that ∀j $D_i \not\equiv D$ • If ∃i $\varepsilon_R(R)_i$=(R,Re) and $\rho_{\exists R}(Re) = \{Re, De_1), \dots, (Re, De_n)\}$ such that ∀j $D \neq De_j$ • if $\iota_R(R)$={(R,(iD_1,iRa_1)),…,(R,(iD_n,iRa_1))} and ∃i such that ∀j $iRa_i \notin$ D

AddRoleValueRestriction

The AddRoleValueRestriction change corresponds to the addition of a value restriction to a role R. These change unsatisfiability constraints are also applied on R equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R Instance I	$\rho_R(R)_i$=(R,I_i)	• If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and $\rho_R(R) = \{(R, I_1), \dots, (R, I_n)\}$ and ∃i $\rho_R(R_i) = \{(R_i, I_1), \dots, (R_i, I_n)\}$ such that ∀j $I_{ij} \neq I_j$ • If ∃i $\varepsilon_R(R)_i$=(R,Re) and $\rho_R(R) = \{(R, I_1), \dots, (R, I_n)\}$and $\rho_R(Re) = \{Re, Ie_1), \dots, (Re, Ie_n)\}$ such that ∀j $I_j \neq Ie_j$ • if $\iota_R(R)$={(R,(iD_1,iRa_1)),…,(R,(iD_n,iRa_1))} and $\rho_R(R) = \{(R, I_1), \dots, (R, I_n)\}$and ∃i such that ∀j $I_i \neq iRa_j$

AddRoleMinCardinalityRestriction

The AddRoleMinCardinalityRestriction change corresponds to the addition of a minimal cardinality restriction to a role R. These change unsatisfiability constraints are also applied on R equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R, Integer n	$\text{minCard}_R(R)$=(R,n)	• If $\leq_R (R) = \{(R, R_1), \dots, (R, R_n)\}$ and $\text{minCard}_R(R) = (R, n)$ and ∃i such that $\text{minCard}_R(R_i) = (R, m)$ with $n \neq m$ • If $\varepsilon_R(R)_i$=(R,Re) and $\text{minCard}_R(R) = (R, n)$ and $\text{minCard}_R(Re) = (Re, m)$ such that $n \neq m$ • If $\kappa_R(R)_i$={functional} and $\text{minCard}_R(R) = (R, n)$ such that $n \neq 1$ • If $\kappa_R(R)_i$={symmetric} and $\text{minCard}_R(R) = (R, n)$ and $-_R(R)_i$=(R,$R\text{-}_i$) and ∃i such that $\text{minCard}_R(R\text{-}_i)$=($R\text{-}_i$,m) with $n \neq m$ • If $\text{minCard}_R(R) = (R, n)$ and $\text{maxCard}_R(R) = (R, m)$ such that n>m • If $\text{minCard}_R(R) = (R, n)$ and $\iota_R(R)$={(R,(iD_1,iRa_1)),…, (R,(iD_1,iRa_j)),…, ((R,(iD_m,iRa_k))} and ∃i such that $\iota_R(R)_i$={(R,(iD_i,iRa_1)), …, (R,(iD_i,iRa_l)) with l<n

AddRoleMaxCardinalityRestriction

The AddRoleMaxCardinalityRestriction change corresponds to the addition of a maximal cardinality restriction to a role R. These change unsatisfiability constraints are also applied on R equivalent roles.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R, Integer n	$\text{maxCard}_R(R)=(R,n)$	• If $\leq_R(R)=\{(R,R_1),\dots,(R,R_n)\}$ and $\text{maxCard}_R(R)=(R,n)$ and $\exists i$ such that $\text{maxCard}_R(R_i)=(R,m)$ with $n \neq m$ • If $\varepsilon_R(R)_i=(R,Re)$ and $\text{maxCard}_R(R)=(R,n)$ and $\text{maxCard}_R(Re)=(Re,m)$ such that $n\neq m$ • If $\kappa_R(R)_i=\{\text{functional}\}$ and $\text{maxCard}_R(R)=(R,n)$ such that $n\neq 1$ • If $\kappa_R(R)_i=\{\text{symmetric}\}$ and $\text{maxCard}_R(R)=(R,n)$ and $-_R(R)_i=(R,R\text{-}_i)$ and $\exists i$ such that $\text{maxCard}_R(R\text{-}_i)=(R\text{-}_i,m)$ with $n\neq m$ • If $\text{maxCard}_R(R)=(R,n)$ and $\text{minCard}_R(R)=(R,m)$ such that $n<m$ • If $\text{maxCard}_R(R)=(R,n)$ and $\iota_R(R)=\{(R,(iD_1,iRa_1)),\dots,(R,(iD_1,iRa_j)),\dots,((R,(iD_m,iRa_k))\}$ and $\exists i$ such that $\iota_R(R)_i=\{(R,(iD_i,iRa_1)),\dots,(R,(iD_i,iRa_l))$ with $l>n$

AddEquivalentAttribute

The AddEquivalentAttribute change corresponds to the addition of an equivalence constraint between two attributes A1 and A2. These change unsatisfiability constraints are also applied on A1 and A2 equivalent attributes and subattributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attributes A1, A2	$\varepsilon_A(A1)_i=(A1,A2)$	16. if $\leq_A(A1)=\{(A1,A1_1),\dots,(A1,A1_n)\}$ and $\leq_A(A2)=\{(A2,A2_1),\dots,(A2,A2_n)\}$and $\exists i, j$ such that $\sigma_A(A1_i)=\{(A1_i,(D1_1,T1_1)),\dots,(A1_i,(D1_n,T1_m))\}$ and $\sigma_A(A2_i)=\{(A2_i,(D2_1,T2_1)),\dots,(A2_i,(D2_n,T2_m))\}$ and $\forall k \exists l$, x, y such that $(D1_k,T1_x)\neq(D2_l,T2_y)$ • If $\kappa_A(A1)=\{(A1,KA1_1),\dots,(A1,KA1_n)\}$ and $\kappa_A(A2)=\{(A2,KA2_1),\dots,(A2,KA2_n)\}$ such that $\forall i \exists j$ $KA1_i \neq KA2_j$ • If $\rho_{\exists A}(A1)=\{(A1,(D1_1,T1_1)),\dots,(A1,(D1_n,T1_m))\}$ and $\rho_{\exists A}(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_m))\}$ and $\forall i \exists j$, x, y such that $(D1_i,T1_x)\neq(D2_j,T2_y)$ • If $\rho_{\forall A}(A1)=\{(A1,(D1_1,T1_1)),\dots,(A1,(D1_n,T1_n))\}$ and $\rho_{\forall A}(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_n))\}$ and $\forall i \exists j$, x, y such that $(D1_i,T1_x)\neq(D2_j,T2_y)$ • If $\rho_A(A1)=\{(A1,(D1_1,V1_1)),\dots,(A1,(D1_n,v1_n))\}$ and $\rho_A(A2)=\{(A2,(D2_1,V2_1)),\dots,(A2,(D2_n,V2_n))\}$ and $\forall i \exists j$, x, y such that $(D1_i,V1_x)\neq(D2_j,V2_y)$ • *If* $\sigma_A(A1)=\{(A1,(D1_1,T1_1)),\dots,(A1,(D1_n,T1_n))\}$ and $\sigma_A(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_n))\}$ and $\forall i \exists j$, x, y such that $(D1_i,T1_x)\neq(D2_j,T2_y)$ • if $\iota_A(A1)=\{(A1,(ID1_1,V1_1)),\dots,(A1,(ID1_n,V1_1))\}$ and $\iota_A(A2)=\{(A2,(ID2_1,V2_1)),\dots,(A2,(ID2_n,V2_1))\}$ and $\forall i\ j,x,y$ such that $(ID1_i,V1_x)\neq(ID2_j,V2_y)$

AddSubAttribute

The AddSubAttribute change corresponds to the addition of an inclusion between two attributes A1 and A2. These change unsatisfiability constraints are also applied on A1 and A2 equivalent attributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attributes A1, A2	$\leq_A(A1)_i=(A1,A2)$	• if $\exists i \leq_A(A2)_i=(A2,A2_i)$ and $\kappa_A(A1)=\{(A1,KA1_1),\dots,(A1,K1_n)\}$ and $\kappa_A(A_i)=\{(A_i,KA_{i1}),\dots,(A_i,KA_{in})\}$ such that $\forall j \exists k$ $KA1_j \neq KA_{ik}$ • If $\varepsilon_A(A2)=(A2,Ae)$ and $\kappa_A(A1)=\{(A1,KA1_1),\dots,(A1,K1_n)\}$ and $\kappa_A(Ae)=\{(Ae,KAe_1),\dots,(Ae,KAe_n)\}$ such that $\forall j \exists k$ $KA1_j \neq KAe_k$ • $\rho_{\exists A}(A1)=\{(A1,(A1_1,T1_1)),\dots,(A1,(D1_n,T1_1))\}$ and $\rho_{\exists A}(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_1))\}$ and $\forall i \not\exists j \leq_C(D1_i)_j=(D1_i,D2_j)$ and $T2_j=T1_i$ • $\rho_{\forall A}(A1)=\{(A1,(D1_1,T1_1)),\dots,(A1,(D1_n,T1_1))\}$ and $\rho_{\forall A}(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_1))\}$ $\forall i \not\exists j \leq_C(D1_i)_j=(D1_i,D2_j)$ and $T2_j=T1_i$ • $\rho_A(A1)=\{(A1,(D1_1,V1_1)),\dots,(A1,(D1_n,V1_1))\}$ and $\rho_A(A2)=\{(A2,(D2_1,V2_1)),\dots,(A2,(D2_n,V2_1))\}$ and $\forall i \not\exists j \leq_C(D1_i)_j=(D1_i,D2_j)$ and $\varepsilon_I(V1_i)=(V1_i,V2_j)$ • $\sigma_A(A1)=\{(A1,(D1_1,T1_1)),\dots,(R1,(D1_n,T1_1))\}$ and $\sigma_A(A2)=\{(A2,(D2_1,T2_1)),\dots,(A2,(D2_n,T2_1))\}$ and $\forall i \not\exists j \leq_C(D1_i)_j=(D1_i,D2_j)$ and $T2_j=T1_i$ • if $\iota_A(A2)=\{(A2,(ID2_1,V2_1)),\dots,(A2,(ID2_n,V2_n))\}$ and $\sigma_A(A1)=\{(A1,(D1_1,T1_1)),\dots,(R1,(D1_n,T1_1))\}$ and $\forall i \not\exists j$ $\iota_C(D1_i)=(D1_i,ID2_j)$ and $\iota_T(T1_i)=(T1_i,V2_j)$

AddAttributeSignature

The AddAttributeSignature change corresponds to the addition of a signature to an attribute A. This change unsatisfiability constraints also apply on A equivalent attributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A Concept D Datatype T	$\sigma_A(A)_i=(A,(D,T))$	• if $\leq_A(A)=\{(A,A_1),\dots,(A,A_n)\}$ and and $\exists i \ \exists j$ $\sigma_A(A_j)_k=(A_j(D_k,T_k)$ such that $\not\exists k \leq_C(D)_k=(D,D_j)$ • if $\varepsilon_A(A)=\{(A,Ae_1),\dots,(A,Ae_n)\}$ and $\exists i \exists j$ $\sigma_A(Ae_j)_k=(Ae_j(De_k,Te_k))$ with

∄k $\varepsilon_C(D)_k(D,De_j)$
- if $\varepsilon_A(A) = \{(A, Ae_1), \dots, (A, Ae_n)\}$ and ∃i∃j $\sigma_A(Ae_i)_k = (Ae_j(De_k,Te_k))$ with $T \neq Te_j$
- if ∃i $\iota_A(A)_i = (A,(ID_i,V_i))$ such that ∄j $\iota_C(D)_j = (D,ID_i)$
- if ∃i $\iota_A(A)_i = (A,(ID_i,V_i))$ such that ∄k $\iota_C(T)_k = (T,V_i)$

AddAttributeFunctionalProperty

The AddAttributeFunctionalProperty change corresponds to the addition of a functional property to an attribute A. These change unsatisfiability constraints are also applied on A equivalent attributes and subattributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A	$\kappa_A(A)_i$=(A,Functional)	• if $\leq_A (A) = \{(A, A_1), \dots, (A, A_n)\}$ and ∀i∄j such that $\kappa_A(A_i)_j$=(A_i,Functional) • If ∃i such that $\varepsilon_A(A)_i$=Ae and ∄j $\kappa_A(Ae)_j$=(Ae,Functional) • if $\iota_A(A)$={(A,(ID_1,V_1)),…,(A,(ID_n,V_1))} and ∃i, ∃ j such that $\varepsilon_A(ID_i) = (ID_i, ID_j)$ and $V_i \neq V_j$

AddAttributeUniversalRestriction

The AddAttributeUniversalRestriction change corresponds to the addition of an universal restriction to an attribute A. This change unsatisfiability constraints also apply on A equivalent attributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A Datatype T	$\rho_{\forall A}(A)_i$=(A,T)	• If $\leq_A (A) = \{(A, A_1), \dots, (A, A_n)\}$ and ∃i $\rho_{\forall A}(A_i) = \{(A_i, T_1), \dots, (A_i, T_n)\}$ such that ∄j $\leq_T (T)$=(T, T_j) • If ∃i $\varepsilon_A(A)_i$=Ae and $\rho_{\forall A}(Ae) = \{Ae, Te_1), \dots, (Ae, Te_n)\}$ such that ∀j $T_j \neq Te_j$ • if $\iota_A(A)$={(A,(ID_1,V_1))},…,(A,(ID_n,V_1))} and ∄j $\iota_T(T)_i$=(T,V_i)

AddAttributeExistentialRestriction

The AddAttributeExistentialRestriction change corresponds to the addition of an existential restriction to an attribute A. This change unsatisfiability constraints also apply on A equivalent attributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A Datatype T	$\rho_{\exists A}(A)_i$=(A,T)	• If $\leq_A (A) = \{(A, A_1), \dots, (A, A_n)\}$ and ∃i $\rho_{\exists A}(A_i) = \{(A_i, T_{i1}), \dots, (A_i, T_{in})\}$ such that ∄j $\leq_T (T)$=(T,T_{ij}) • If ∃i $\varepsilon_A(A)_i$=(A,Ae) and $\rho_{\exists A}(Ae) = \{(Ae, Te_1), \dots, (Ae, Te_n)\}$ such that ∀j $T \neq Te_j$ • if $\iota_A(A)$={(A,(ID_1,V_1))},…,(A,(ID_n,V_1))} and ∄i $\iota_T(T)_i$= (T,V_i)

AddAttributeValueRestriction

The AddAttributeValueRestriction change corresponds to the addition of a value restriction to an attribute A. This change unsatisfiability constraints also apply on A equivalent attributes.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A DataValue V	$\rho_A(A)_i$=(A,V)	• If $\leq_A (A) = \{(A, A_1), \dots, (A, A_n)\}$ and ∃i $\rho_A(A_i) = \{(A_i, V_{i1}), \dots, (A_i, V_{in})\}$ such that ∀j $V_{ij} \neq V$ • If ∃i $\varepsilon_A(A)_i$=(A,Ae) and $\rho_A(Ae) = \{Ae, Ve_1), \dots, (Ae, Ve_n)\}$ such that ∀j $V \neq Ve_j$ • if $\iota_A(A)$={(A,(ID_1,V_1))},…,(A,(ID_n,V_1))} and ∃i such that $V_i \neq V$

AddSameInstance

The AddSameInstance change corresponds to the addition of an instance equivalence constraint to an instance I.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Instance I1, I2	$\varepsilon_I(I1)_i$=(I1,I2)	• if ∃i $\iota_C(D1)_i$=(D1,I1) and ∃j $\iota_C(D2)_j$=(D2,I2) and $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$ and ∄k such that $\leq_C (D2) = (D2, D1_k)$ • if $\partial_I(I1)$=(I1,I2) • if ∃i $\iota_C(D1)_i$=(D1,I1) and ∃j $\iota_C(D2)_j$=(D2,I2) and ∄k $\varepsilon_C(D1)_k$=(D1,D2)

AddDifferentInstance

The AddDifferentInstance change corresponds to the addition of an instance difference constraint to an instance I.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Instance I1, I2	∂_I(I1)=(I1,I2)	• if ∃i $\iota_C(D1)_i$=(D1,I1) and ∃j $\iota_C(D2)_j$=(D2,I2) and $\leq_C (D1) = \{(D1, D1_1), \dots, (D1, D1_n)\}$ and ∃k such that $\leq_C (D2) = (D2, D1_k)$ • if ε_I(I1)=(I1,I2) • if ∃i $\iota_C(D1)_i$=(D1,I1) and ∃j $\iota_C(D2)_j$=(D2,I2) and ∃k $\varepsilon_I(I1)_k$=(I1,I2)

AddConceptInstanciation

The AddConceptInstanciation change corresponds to the addition of a concept instanciation. This change unsatisfiability constraints also apply on I equivalent instances and D superconcepts.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Concept D, Instance I	ι_C(D)=(D,I)	• if ∃i $\iota_C(De)_i$=(De,I) and ∄j $\varepsilon_C(De)_i$= (De,D) • if ∃i $\iota_R(R)_i$=(R,(I, IRa$_i$)) and ∀j $\sigma_R(R)_j$=(R,(D$_j$,Ra$_j$)) with ∄k ε_C(D$_j$)$_k$=(D$_j$,D) • if ∃i $\iota_R(R)_i$=(R,(ID$_i$,IRa)) and ∀j $\sigma_R(R)_j$=(R,(D$_j$,Ra$_j$)) and ∄k ε_C(Ra$_j$)$_k$=(Ra$_j$,D) • if ∃i $\iota_A(A)_i$=(A,(I, V$_i$)) and ∀j $\sigma_A(A)_j$=(A,(D$_j$,T$_j$)) with ∄k ε_C(D$_j$)$_k$=(D$_j$,D) • if ∃i $\sqcap_{iC}(De)_i$=(De,{I$_1$,…I$_n$}) and ∃j ε_I (I$_j$)=(I$_j$,I) such that ∄k ε_C (D)$_k$ = (D,De) and ∄l such that $\leq_C$(D)$_l$=(D,De) • if ∃i $\sqcup_{iC}(De)_i$=(De,{I$_1$,… I$_n$}) and ∃j ε_I (I$_j$)=(I$_j$,I) such that ∄k ε_C (D)$_k$ = (D,De) and ∄l such that $\leq_C$(D)$_l$=(D,De) • if ε_I(I)=(I,I1) and ∃i $\iota_C(De)_i$=(De,I1) and ∄j ε_C(D)$_j$=(D,De) • if ε_I(I1)=(I1,I) and ∃i $\iota_C(De)_i$=(De,I1) and ∄j ε_C(D)$_j$=(D,De) • if ∂_I(I)=(I,I1) and ∃i $\iota_C(De)_i$=(De,I1) and ∃j ε_C(D)$_j$=(D,De) • if ∂_I(I1)=(I1,I) and ∃i $\iota_C(De)_i$=(De,I1) and ∃j ε_C(D)$_j$=(D,De)

AddDatatypeInstanciation

The AddDatatypeInstanciation change corresponds to the addition of a datatype instanciation.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Datatype T, Datavalue V	ι_T(T)$_i$=(T,V)	• if ∃i $\iota_T(T1)_i$=(T1,V) and T1≠T • if ∃i $\iota_A(A)_i$=(A,(I$_i$, V)) and ∀j $\sigma_A(A)_j$=(A,(D$_j$,T$_j$)) with T$_j$ ≠T • if ∃i $\sqcup_V (T1)_i$=(T1,{ V$_1$,… V$_n$}) and ∃j V=V$_j$ and T1≠T and ∀k $\leq_T$(T1)$_k$ ≠ (T1,T)

AddRoleInstanciation

The AddRoleInstanciation change corresponds to the addition of a role instanciation.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Role R, Instance I1, I2	ι_R(R)$_i$=(R(I1,I2))	• if ∃i $\iota_C(D)_i$=(D,I1) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∄j ε_C(D)$_j$=(D,D$_j$)and ∄k $\leq_C$(D)$_k$=(D,D$_k$) • if ∃i $\iota_C(Ra)_i$=(Ra,I2) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∄j ε_C(Ra)$_j$=(Ra,Ra$_j$)and ∄k such that $\leq_C$(Ra)$_k$=(Ra,Ra$_k$) • if ∃i $\iota_C(D)_i$=(D,I1) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∃j ε_C(D)$_j$=(D,D$_j$) and ∃k $\iota_C(Ra)_k$=(Ra,I2) and ∄l ε_C(Ra)$_l$=(Ra,Ra$_j$) • if ∃i $\iota_C(D)_i$=(D,I1) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∃j $\leq_C$(D)$_j$= (D, D$_j$) and ∃k $\iota_C(Ra)_k$=(Ra,I2) and ∄l ε_C(Ra)$_l$=(Ra,Ra$_l$) • if ∃i $\iota_C(Ra)_i$=(Ra,I2) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∃j ε_C(Ra)$_j$=(Ra,Ra$_j$) and ∃k $\iota_C(D)_k$=(D,I1) and ∀l ε_C(D)$_l$=(D,D$_l$) if ∃i $\iota_C(Ra)_i$=(Ra,I2) and $\sigma_R(R)$ ={(R,(D$_1$,Ra$_1$)),…, (R,(D$_n$,Ra$_m$))} and ∃j $\leq_C$(Ra)$_j$= (Ra, Raj)] and ∃k $\iota_C(D)_k$=(D,I1) and ∀l ε_C(D)$_l$=(D,D$_l$) • if minCard$_R(R)$=(R,n) and ∃i $\iota_R(R)_i$=(R,(I1, IRa$_1$)) ,…, R,(I1, IRa$_m$)) and m<n • if maxCard$_R(R)$=(R,n) and ∃i $\iota_R(R)_i$=(R,(I1, IRa$_1$) ,…, R,(I1, IRa$_m$)) and m>n • if $\rho_R(R)$={(R,I$_1$), …, (R,I$_n$)} and ∄i ε_I(I2)$_k$=(I2, I$_i$) • if ∃i $\kappa_R(R)_i$=(R,Symmetric) and ∃j $\iota_C(D)_j$=(D,I1) and ∃k $\iota_C(Ra)_k$=(Ra,I2) and ∄l ε_C(D)$_l$=(D, Ra) • if ∃i $\kappa_R(R)_i$=(R,Functional) and ∃j $\iota_R(R)_j$=(R,(I1, IRa$_1$) ,…, (R,(I1, IRa$_m$)) and m>1

AddAttributeInstanciation

The AddAttributeInstanciation change corresponds to the addition of an attribute instanciation.

Entities	$\mathcal{SHOIN}(\mathcal{D})$ Model Axiom	Unsatisfiability Constraints
Attribute A, Instance I, Datavalue V	$\iota_A(A)_i$=(A(I,V))	• if ∃i $\iota_C(D)_i$=(D,I) and $\sigma_A(A)$ ={(A,(D_1,T_1)),..., (R,(D_n,T_m))} and ∄j $\varepsilon_C(D)_j$=(D,D_j) and ∄k $\leq_C(D)_k$=(D,D_k) • if ∃i $\iota_T(T)_i$=(T,V) and $\sigma_A(A)$ ={(A,(D_1,T_1)), ..., (A,(D_n,T_n))} and ∀j T≠ T_j and ∄k $\leq_C(T)_k$=(T,T_k) • if ∃i $\iota_C(D)_i$=(D,I) and $\sigma_A(A)$ ={(A,(D_1,T_1)),..., (A,(D_n,T_n))} and ∃j $\varepsilon_C(D)_j$=(D,D_j) and ∃l $\iota_T(T)_l$=(T,V) and ∀m T≠ T_m • if ∃i $\iota_C(D)_i$=(D,I) and $\sigma_A(A)$ ={(A,(D_1,T_1)),..., (A,(D_n,T_n))} and ∃j $\leq_C(D_j)_j$= (D_j, D) and ∃l $\iota_T(T)_l$=(T,V) and ∀m T≠ T_m • if ∃i $\iota_T(T)_i$=(T,V) and $\sigma_A(A)$ ={(A,(D_1,T_1)),..., (A,(D_n,T_n))} and ∃j T=T_j and ∃l $\iota_C(D)_l$=(D,I) and ∀m D≠ D_m • if ∃i $\iota_T(T)_i$=(T,V) and $\sigma_A(A)$ ={(A,(D_1,T_1)),..., (A,(D_n,T_n))} and ∃j T=T_j and ∃j $\leq_T(T_j)_j$= (T_j, T) and ∃l $\iota_C(D)_l$=(D,I) and ∀m D≠ D_m • if $\rho_A(A)$={(A,V_1), ..., (A,V_n)} and ∀i V≠V_i • if ∃i $\kappa_A(A)_i$=(A,Functional) and ∃i $\iota_A(A)_i$=(A,(I, V_1) ,..., (A,(I, V_m)) and m>1

Notes

SPIM

■ École doctorale SPIM - Université de Bourgogne/UFR ST BP 47870 F - 21078 Dijon cedex
■ tél. +33 (0)3 80 39 59 10 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UNIVERSITÉ DE BOURGOGNE