Towards Methodological Principles for Ontology Engineering

A thesis submitted by

Mustafa Jarrar

For the degree of

Doctor of Philosophy

Vrije Universiteit Brussel
Faculty of science
May 2005

Promoter: Professor Dr. Robert Meersman

Dr. Mustafa Jarrar
Senior Research Scientist
Marie Curie Postdoc Fellow

University of Cyprus
Phone: 357 22 892.676 | Fax: +357 22 892.701
mustafa (at) jarrar.info
www.jarrar.info

Downloads and pages related to this thesis:

- Thesis: http://www.jarrar info/phd-thesis
- DogmaModeler: http://www.jarrar.info/Dogmamodeler/
- ORM Markup Language (Ver 3.0): http://www.jarrar.info/publications/index.htm#[J07a]
- ORM Verbalizor: http://www.jarrar.info/orm/verbalization/
- The Customer Complaints ontology: http://www.jarrar.info/CContology
- Blog: http://mjarrar.blogspot.com/
- Related publications: http://www.jarrar.info/publications/

-D

-D

-D

Abstract

The Internet and other open connectivity environments create a strong demand for the sharing of data semantics. Emerging ontologies are increasingly becoming essential for computer science applications. Organizations are looking towards them as vital machine-processable semantics for many application areas. An ontology in general, is an agreed understanding (i.e. semantics) of a certain domain, axiomatized and represented formally as logical theory in a computer resource. By sharing an ontology, autonomous and distributed applications can meaningfully communicate to exchange data and make transactions interoperate independently of their internal technologies.

The main goal of this thesis is to present methodological principles for ontology engineering to guide ontology builders towards building ontologies that are both highly reusable and usable, easier to build, and smoother to maintain.

First, we investigate three foundational challenges in ontology engineering (namely, ontology reusability, ontology application-independence, and ontology evolution). Based on these challenges, we derive six ontology-engineering requirements. Fulfilling these requirements is the goal and motivation of our methodological principles.

Second, we present two methodological principles for ontology engineering: 1) ontology double articulation, and 2) ontology modularization. The double articulation principle suggests that an ontology be built as separate domain axiomatizations and application axiomatizations. While a domain axiomatization focuses on the characterization of the intended meaning (i.e. intended models) of a vocabulary at the domain level, application axiomatizations mainly focus

-D

on the usability of this vocabulary according to certain application/usability perspectives. An application axiomatization is intended to specify the legal models (a subset of the intended models) of the application(s)' interest. The modularization principle suggests that application axiomatizations be built in a modular manner. Axiomatizations should be developed as a set of small modules and later composed to form, and be used as, one modular axiomatization. We define a composition operator for automatic module composition. It combines all axioms introduced in the composed modules.

Third, to illustrate the implementation of our methodological principles, we develop a conceptual markup language called ORM-ML, an ontology engineering tool prototype called DogmaModeler and a customer complaint ontology that serves as a real-life case study.

This research is a contribution to the DOGMA research project, which is a research framework for modeling, engineering, and deploying ontologies. In addition, we find we have benefited enormously from our participation in several European projects. It was through the CCFORM project (discussed extensively in chapter 7) that we were able to test and debug many ideas that resulted in this thesis. The Network of Excellence KnowledgeWeb has also proved to be a fruitful brainstorming environment that has undoubtedly improved the quality of the analyses performed and the results obtained.

To my Parents

To my country Palestine

To all donors in the world

-D

-D

Acknowledgments

At last, I am very glad to write this page. The relief and accomplishment I feel in having come to this stage comes with a deep sense of indebtedness to the help, support, and inspiration of the many people to whom the thesis owes its existence.

First of all, I wish to express my sincere gratitude to Professor Robert Meersman, the promoter and the friend, who guided this work and helped whenever I was in need. Robert's direction and support have been invaluable, not only in science but also in life experience. I was blessed in the last five years, with his guidance, encouragement, tolerance, freedom, trust, hospitality, and his friendship.

To the members of the jury - Professor Dr. Fausto Giunchiglia, Professor Dr. Esteban Zimanyi, Professor Dr. Dirk Vermeir, and the president of jury, Professor Dr. Luc Steels - I am most grateful for the precious time you all devoted to reading this. It is my honor and I thank you for the advice and the constructive criticism that contributed substantially to bringing the original conception to this final stage.

I wish to express my debt to all present and former colleagues in STARLab who have provided me with inspiration, advice, and encouragement, and who have so generously shared their knowledge and technical expertise with me. Especially, I am indebted to my colleague Andriy Lisovoy, who helped me in the implementation of DogmaModeler. Andriy is more than a colleague: I thank him also for the richness he brought to my social and especially for the inspiring coffee breaks that we spent together. I wish him great success in his PhD and in his life in general.

I would like to thank also, Stijn Heymans for reviewing chapter 5. His discussion and suggestions have influenced my work significantly.

I am in debt to many other colleagues for the useful discussions, we had on different occasions - of which served greatly to influence my research. In particularly I wish to thank Andreas Persidis, Stefano Spaccapietra, Olga De Troyer, Luk Vervenne, Jan Demey, Nicola Guarino, Enrico Franconi, Jeff Z. Pan, Luciano Serafini, Giancarlo Guizzardi, Paolo Bouquet, Aldo Gangemi, Mohand-Saïd Hacid, Robert Colomb, Rita Temmerman, Werner Ceusters, and Stefano Borgo.

I also gratefully acknowledge the financial support I received from the BTC-CTB, as part of a mixed scholarship offered under the cooperation program between Belgium and Palestine.

Finally, I dedicate this work to my parents, my sons, my country, Palestine, and to all donors in the world.

Contents

-D

List of Figures

List of Tables

Chapter 1

Introduction and Overview

"The process of building or engineering ontologies for use in information systems remains an arcane art form, which must become a rigorous engineering discipline."

- (Guarino et al., [GW02])

The central goal of this thesis is to develop methodological principles for ontology engineering. We briefly outline the scope and motivation of the thesis in section 1.1. In section 1.2, we summarize the main goals and contributions of the thesis and in section 1.3, we give an overview of the thesis outline.

1.1 Scope and motivation

The Internet and open connectivity environments create a strong demand for the sharing of data semantics. Emerging ontologies are increasingly becoming essential for computer science applications. Organizations are beginning to view them as useful machine-processable semantics for many application areas. Some examples of such applications are:

- e-commerce content standards [GP03][BCW97][CG01],
- bioinformatics [Gene00] [BBB+98] [KRS+02],
- geographical information systems [F97][FE99][U01][RSV98],
- regulatory and legal information systems [BVW97][GP01][JS03],
- digital libraries [SMD00][W98] [BDMW95],
- e-learning [SKC02][AKS04][VKMND04],
- agent technology [FLS96][TB01][K03],
- database design [G02] and integration [W95][WSW99],
- software engineering [DW00][WF99][M98],
- natural language processing [K96][CC03][BCW02],
- information access and retrieval [GMV99][ACFOH03][AR00],
- the Semantic Web [BF99][M04][GAC+04],
- Web services [BLA+05][NM02],
- etc.

An ontology in general, is a shared understanding (i.e. semantics) of a certain domain, axiomatized and represented formally -as logical theory- in a computer resource. By sharing an ontology, autonomous and distributed applications can meaningfully communicate to exchange data and make transactions interoperate independently of their internal technologies. In this way, heterogeneous and distributed information resources can be integrated and searched through mediators [TSC01] [SOV+02].

In recent years, research on ontologies has turned into an interdisciplinary subject. It combines elements of Philosophy (especially what is now called Analytic Philosophy [S03a]), Linguistics (mainly lexical semantics [KTT03]), Logic (in particular, first-order logic and its derivatives, e.g.

description logic [BCMNP03]), and Computer Science. Within computer science, the research on ontologies emerged “mainly” within two subcommunities: artificial intelligence (among scientists largely committed to building shared knowledge bases) and database (among scientists and members of industry who are largely committed to building conceptual data schemes, also called semantic data models [V82]).

Unlike a conceptual data schema or a “classical” knowledge base that captures semantics for a given enterprise application, the main and fundamental advantage of an ontology is that it captures domain knowledge highly independently of any particular application or task [JDM03]. A consensus on ontological content is the main requirement in ontology modeling, and this is what mainly distinguishes it from conceptual data modeling. Neither an ontology nor its development process is a single person enterprise [KN03].

1.1.1 Foundational challenges in ontology engineering

In this section, we briefly present critical challenges that face the endeavor of the ontology development life cycle. We consider tackling these challenges as the goal of our research.

- Ontology reusability. Reusability implies the maximization of an ontology’s use across different kinds of applications or tasks, i.e. among different purposes [JDM03][JM02a]. The main benefits of ontology reuse are not only savings in time, cost, and efforts, but also an increase in “reliability” [HV93]. A highly reusable ontology gives the indication that it is generally accepted (it fosters trust and consensus). Considering the reusability during the development phase will assist in ensuring that the resulting ontology to be specific for and dependent on certain purposes. The more reusable an ontology is, the more it will be independent from specific needs. This is an essential goal for ontology development methodologies to guide ontology builders towards more reusability

[G97]. The main challenges that hamper ontology reusability are 1) the influence of a specific purpose (what it is made for) on the ontology developer and 2) the difficulty of identifying and isolating the reusable components (i.e. allowing the reuse of the general-purpose parts of an ontology).

- Ontology application/task-independence. Ontologies are supposed to capture semantics at the domain level and be independent of application requirements [G97][CJB99][M99a][JDM03]. One problem that arises when building an ontology is that there will always be intended or expected application requirements "at hand" (i.e. usability perspectives) which influence the independence of ontology axioms. Different usability perspectives (i.e. different purposes of *what an ontology is made for* and *how it will be used*) lead to different or even to conflicting axiomatizations, although these axiomatizations might intuitively be in agreement at the domain level. The more an axiomatization is independent of application perspectives, the less usable it will be. In contrast, the closer an axiomatization is to application perspectives, the less reusable it will be. From a methodological viewpoint, notice that if a methodology emphasizes usability perspectives, or evaluates ontologies based only on how they fulfill specific application requirements, the resultant ontology will be similar to a conceptual data schema (or a classical knowledge base) containing specific - and thus less reusable - knowledge. Likewise, if a methodology emphasizes only on the independence of the knowledge and ignores application perspectives, the resultant ontology will be less *usable*.

- Ontology evolution. The continuous growth and intensive maintenance of emerging ontologies currently (and for the immediately foreseeable future) are serious challenges in the ontology development life cycle [Hj01] [KKOF02] [MMS03].

> Ontologies evolve over time, due to conceptual changes, epistemological changes, scope extensions, mistakes and quality improvements, etc. Such changes have implication for the applications that have committed to a changing ontology. More significantly however, the evolution processes itself becomes more complex in the case of large-scale ontologies. Ontologies are being developed, reviewed, used, and maintained by different people and experts over different times and locations. Thus, we believe that this challenge should not only be tackled through technical or ad hoc solutions, but through an effective foundation of ontology engineering that enables the smooth evolution of ontologies.

Consequently, such challenges imply the importance of *a solid and a principled methodology for ontology engineering that provides guidance for developin*g "true" ontologies with minimum cost, time and effort.

1.1.2 Types of methodologies

According to the guiding scenario that a methodology provides, we distinguish between a stepwise methodology, a modeling methodology, and an engineering methodology[1].

A stepwise methodology divides the ontology development process into a set of phases, and provides a series of steps and guidelines to be followed in each phase. For example, the Methontology [FGJ97] methodology divides the ontology development life cycle into: specification, conceptualization, formalization, implementation, and maintenance. The On-To-Knowledge [S03b] methodology divides it into: feasibility study, kickoff, refinement, evaluation, and applications & evolution. As an analogy, the development process of a software program according to the classical "Waterfall" methodology [R70] is divided into: specification, requirement analysis, design, implementation, and testing.

[1] The goal of this distinction is to motivate and understand the general scope of the thesis.

A modeling methodology is concerned with the formal analysis of a given domain: what kinds of modeling decision need to be made and how these decisions can be evaluated. Such domain analysis (the modeling process) can be performed typically by means of a set of well-defined modeling constructs and primitives, e.g. the notions of concept/class, n-ary relations/roles, functions, properties/attributes, constraint/rule types, etc. As an analogy, the Object Role Modeling ORM [H01], and the Enhanced Entity Relationship EER [EN99] are modeling methodologies for building database schemes. They provide database designers with a set of primitives by which they can be guided to build normalized database schemes. In ORM, for instance, the world can be analyzed and modeled as objects-types playing roles. In addition, ORM supports a rich set of constraint types such as mandatory, uniqueness, subsumption, equality, exclusive, subset, ring, etc., which allow for the focus on the integrity of data models[2]. For ontologies, the OntoClean [GW02] methodology provides a set of metaproperties, such as essence, rigidity, identity, unity, subsumption, instantiation, etc. These metaproperties (as a theoretical tool or methodology) guide ontology builders to focus on and characterize the intended meaning of the properties, classes, and relations that make up an ontology[3].

An engineering methodology is concerned with the design, representation, architecture, and management aspects of ontologies. The questions it seeks to answer include how to enable ontology reusability, usability, maintainability, distributed development, application-independence, scalability, etc. Engineering methodologies are not concerned *directly* with modeling decisions or phases. By way of analogy, in the software

[2] It is perhaps worthwhile to note that ORM derives from NIAM (Natural Language Information Analysis Method), which was explicitly designed to be a stepwise methodology arriving at "semantics" of a business application's data based on natural language communication.

[3] The OntoClean methodology is mainly concerned with the taxonomic structure of an ontology.

development life cycle, the object-oriented paradigm is the basis for an engineering methodology. This paradigm provides guidance for its adopters (software developers) by encapsulating the complexity of each software module, thus making their products (software programs) more reusable, maintainable, and easy to build as it.

Notice that stepwise methodologies usually are invented based on “best practice”, and their guidance cannot easily be formally captured; cf. the *pattern approac*h in software development [A97b]. In comparison, as both modeling and engineering methodologies are usually based on well-articulated principles, they can be called *principled methodologies*. For any kind of methodology, as suggested by Meersman in [JM02a], this should imply *teachability* and *repeatability*. Indeed, a good methodology must be easy to understand and based on broadly accepted principles.

This thesis is concerned with developing two methodological principles for ontology engineering, with the aim of tackling the ontology development challenges[4] recapped above. Our two fundamental methodological principles are “*Ontology Double Articulation*” and *“Ontology Modularization*”.

Although we present a research prototype of an ontology development tool as part of this study (called DogmaModeler, see chapter 6), it is not a goal of our methodological principles to provide technical or *ad hoc* solutions. We attempt to be general enough in describing our methodological principles, so that they can be applied across domains and application scenarios.

For illustration purposes, we have also developed a conceptual markup language (called ORM-ML, see chapter 5) which allows for the marking up and serialization of ORM conceptual diagrams. However, it is not our

[4] Notice that the ontology development challenges presented in this thesis mostly are engineering challenges. See (e.g. [GW00][U96]) to know about some development challenges that concern the modeling and stepwise methodologies.

goal to develop an ontology language, or reasoning primitives and services.

Further discussions on the motivation and the engineering challenges of ontologies will be presented in chapter 2. The next section summarizes the main goals and contributions of the thesis.

1.2 Summary of the main goals and contributions

The central goal of this dissertation is to develop methodological principles for ontology engineering. The main concerns that distinguish our approach are:

1. Maximization of both reusability and usability of ontologies.
2. Easing of the development and the smoothening of the evolution of ontologies.

Because of the nature of the subject, the contributions of this dissertation will cover a fairly broad spectrum of aspects related to ontology engineering. Keeping in mind the central goals stated above, our contributions can be summarized as:

- *Problem specification.* Several challenges in ontology engineering are discussed and clarified. These include, the influence of usability perspectives in ontology engineering, domain axiomatization verses application axiomatization, the importance of reusability, reusability vs. usability of ontologies, ontology evolution and the importance of linguistic terms in ontology engineering, etc.
- *Methodological principles.* We present two methodological principles for ontology engineering: 1) the "*ontology double articulation*" principle that suggests that ontologies be articulated in two parts: domain axiomatizations and application axiomatizations; 2) the "*ontology modularization*" principle

-D

suggests that application axiomatizations be decomposed into a set of smaller, related modules. The main idea of the double articulation principle is to prevent ontology builders from encoding and mixing their application and usability (specific) axioms with domain axioms. While domain axiomatizations are mainly concerned with capturing the "intended meaning" of domain vocabularies, application axiomatizations are mainly concerned with the "usability" of these vocabularies. As a result, we increase both reusability and usability. To represent an ontology according to this principle, we first introduce the notion of ontology base, for capturing domain axiomatizations. Second, we introduce the notion of ontological commitments to capture application axiomatization, by which particular applications commit to a domain axiomatization. The main idea of the modularization principle is to have smaller modules of axiomatizations, which are easier to develop, reuse, replace, and/or maintain, etc.

Remark: Our research on ontology double articulation is based and builds on the research that was originally conducted by Meersman in [M99a][M99b]. In this thesis, we present fundamental modifications, extensions, and implementation to this idea. For example, we provide precise definitions of the double articulation, context, concept, and introduce the notions of domain axiomatization, gloss, upper-forms, application ontological commitments, etc.

This study is a contribution to the DOGMA[5] research project, which is a research framework for modeling, engineering, and deploying ontologies.

[5] DOGMA stands for "Development of Ontology Guided Methodology Approach".

- *Implementation:* ORM-ML, DogmaModeler, and the CCFORM case study.

  ORM-ML: we have defined a conceptual markup language, called ORM-ML, which allows representing ORM conceptual diagrams in an open and textual syntax. By doing this, we enable the reusing of conceptual data modeling methods and tools -mainly ORM- for modeling, representing, visualizing, and verbalizing application axiomatizations [JDM03].

  DogmaModeler: Based on the ideas presented in this thesis, we have developed an ontology engineering tool, called DogmaModeler. It supports among other things: (1) the development, browsing, and management of domain and application axiomatizations, and axiomatization libraries; (2) the modeling of application axiomatizations using the ORM graphical notation, and the automatic generation of the corresponding ORM-ML; (3) the verbalization of application axiomatizations into pseudo natural language (supporting flexible verbalization templates for English, Dutch, Arabic, and Russian, for example) that allows non-experts to check, validate, or build axiomatizations; (4) the automatic composition of axiomatization modules, through a well-defined composition operator; (5) the validation of the syntax and semantics of application axiomatizations; (6) an illustration of the process of incorporating lexical resources in ontology modeling; (7) a simple approach of multilingual lexicalization of ontologies; (8) the automatic mapping of ORM schemes into X-Forms and HTML-Forms; etc.

  CCFORM case study: The methodological principles and their support tool have been successfully applied in a number of national and European projects such as CCFORM, FFPOIROT, SCOP, etc. To end, we report our experience and main achievements in applying our methodological principles and tool in the CCFORM project, for developing a multilingual Customer Complaint ontology (CContology) [JVM03].

1.3 Thesis outline and structural overview

The thesis is organized in four main parts. *We* specify the problem, propose a solution, and show an implementation of this solution before concluding appropriately.

Part I Problem Specification

Chapter 2 *(Problem specification).* In this chapter we present an extended motivation for the goals of this thesis. We discuss and specify several challenges in ontology engineering. We clarify and define some terminology used in this thesis.

Part II: Methodological Principles

Chapter 3 *(Ontology Double Articulation).* In this chapter, we discuss the "Ontology Double Articulation" methodological principle. We examine the general properties of domain axiomatization verses application axiomatization. We introduce the notion of an ontology base, the notion of an ontological commitment; and show how particular applications commit to the ontology base through ontological commitment(s). The importance of lexical resources in ontology engineering are discussed and incorporated.

Chapter 4 *(Ontology Modularization).* This chapter introduces the "Ontology Modularization" methodological principle. We first present its advantages (e.g. reusability, maintainability, distributed development, etc.). Then we introduce and discuss a set of criterion, which are necessary for achieving an effective modularization. We define a composition operator for composing axiomatization modules.

At the end of this chapter, we present an algorithm for composing ORM schemes (seen as application axiomatization modules).

Part III: Implementation Aspects and Case Study

Chapter 5 *(ORM Markup Language).* In this chapter we define the ORM Markup Language. The motivation for choosing ORM for modeling and representing application axiomatizations is explained.

Chapter 6 *(DogmaModeler Ontology Engineering Tool).* We present the software that we have built to demonstrate the implementation of the two methodological principles. The functionalities supported in DogmaModeler are also discussed.

Chapter 7 *(CCFORM Case Study).* In this chapter, we present a case study of the development of a customer complaint ontology using our methodological principals and the DogmaModeler tool. This ontology itself and the lessons we learnt in applying our methodological principles and tool will be presented and discussed.

Part IV: Conclusions

Chapter 8 *(Conclusions and Future Work).* This chapter summarizes the main ideas of this thesis, and suggests directions for future work.

Appendices: Appendix A lists the XML Schema of the ORM markup language. Appendix B lists the DogmaModeler ontology Metadata, An XML-Schema of the ORM-ML graphical style sheets, and 5 ORM Verbalization Templates. Appendix C lists the Customer Complaint ontology (CCglossary, CC lexons, and seven application axiomatization modules). Finally, appendix D presents a glossary of the terminology that we often use in this thesis.

Part I

Problem specification: Fundamental challenges in ontology engineering

"Semantics is a grand challenge for the current generation of computer technology"

-(David Embley, [E05])

-D

-D

Chapter 2

Fundamental Challenges in Ontology Engineering

"The most important task for the new information systems ontology pertains to what we might call the Database Tower of Babel problem. Different groups of data- and knowledge-base system designers have for historical and cultural and linguistic reasons their own idiosyncratic terms and concepts by means of which they build frameworks for information representation. Different databases may use identical labels but with different meanings; alternatively the same meaning may be expressed via different names. As ever more diverse groups are involved in sharing and translating ever more diverse varieties of information, the problems standing in the way of putting such information together within a larger system increase geometrically."

-(Barry Smith, [S02])

This chapter presents an extended analysis of the goals of this thesis and the motivation driving this endeavor. We investigate and specify several challenges in ontology engineering. Section 2.1 discusses the significance, and challenges of ontology reusability. In section 2.2, we introduce and discuss the most challenging issue in ontology engineering: the application-independence of ontologies. In section 2.3, we clarify some ontology evolution challenges. To end, section 2.4 draws some conclusions and derives the main *ontology engineering requirements*.

2.1 Ontology reusability

Although the role of ontology in information systems is well appreciated in the literature, little attention has been given to research on ontology reusability. Approaches to ontology reusability remain ad hoc. The aim of this section is to discuss what ontology reusability means, the key benefits of reuse, and the main challenges that hamper ontology reusability.

Reusability is one of the most significant aspects in engineering and manufacturing in general. For example, realizing the value of this, software engineers have developed libraries of software routines that are common to different programs to save themselves from having to recode the same routines time and again. In the problem-solving research[6], the importance and techniques of knowledge reusability have been researched to improve the reusability of “problem solving methods” [R00]. Several researchers (e.g. Chandrasekaran and Johnson [CJ93], Clancey [C92], or Swartout and Moore [SM93]) proposed the idea of structuring knowledge into different levels of abstraction. Steels in [S93] proposed a componential framework that decomposes knowledge into reusable components. Many believed that building large knowledge bases would only be possible if efforts are combined (Neches et al. in [PFP+92]). A unified framework to enable and maximize knowledge reusability is advisable.

Supporting and enabling knowledge reusability is an important goal of building ontologies ([UG96] [GPB99] [G95]). *Notice that ontology usability is subtly different from ontology reusability.* Increasing the reusability of knowledge implies the maximization of its usage *among several kinds* of tasks. Increasing ontology usability could just mean maximizing the number of different applications using an ontology for the

[6] This research area was -active in the 80s- focusing on the development of the so-called the next generation of expert systems.

same kind of task[7]. The intended use of the term 'task', in this thesis, is related and limited to the inferential knowledge that is required to describe a task to be performed. It does not describe dynamic or temporal aspects[8]. An application may perform one or more kinds of tasks. In this thesis, the term task is often interchanged with the 'application' that performs one *kind* of task. We sometimes use the term *generic task* to refer to a highly reusable task.

2.1.1 Significance of ontology reusability

The main benefits of ontology reuse are:

- *Savings in time, cost, and efforts*. Instead of constructing an ontology from scratch and repeating the efforts that have already been spent elsewhere to capture and creating the same knowledge, one may reuse an existing ontology or some parts of it[9]. This implies the construction of sharable ontology libraries, such that one can easily search, identify and reuse ontology modules that fit his/her purposes.
- *Increasing reliability* [HV93]. A reusable ontology gives indication that it is approved and generally accepted (i.e. trust and consensus)[10].

[7] For example, compare a Bibliography ontology used by 1000 applications performing the same kind task (e.g. bookselling) with another ontology (of the same subject-matter) used by 100 applications performing different kinds of tasks (e.g. bookselling, borrowing, publishing, etc.). While the former is highly used, the latter is highly reused.
[8] For example, "online bookselling" is a task that can be described by a static knowledge elements or propositions such as: IsA(Book, Product), PublishedBy(Book, Publisher), ValuatedBy(Book, Price), RequestOf(Order, Book), Issues(Customer, Order), SetteledVia(Order, Payment-method), etc.
[9] For example, suppose one wishes to build an ontology of Online Bookstores, he/she may reuse several parts from other existing ontologies of e.g. Customers, Order, Payment-methods, Shipping, etc. which might be developed for and deployed in other application scenarios.
[10] For example, suppose an ontology of payment methods is used in 1000 application scenarios and another ontology of the same subject matter is used only in 3 scenarios.

- They constitute an important *quality factor*. Taking reusability into account during the development phase helps avoid that the resulting ontology to be specific for and dependent on certain purposes (i.e. "requirements at hand"). Pursuing ontology reusability, in the early development phases, will help prevent the ontology from reflecting a particular data model or from being suitable only for one application, etc.

2.1.2 Reusability challenges

In the following, we discuss the *fundamental challenges* that hamper ontology reusability.

The main concern that restricts ontology reuse is *the dependency on the purpose that an ontology is made for*. Although ontologies are intended to capture knowledge at the *domain level*[11], the axiomatization of knowledge can be noticeably influenced by the purpose that this knowledge is made for and how it will be used. In other words, when axiomatizing a domain, several kinds of usability perspectives are usually taken into account (e.g. granularity, scope and relevancy, reasoning/computational scenario, etc.). Thus, when using knowledge for a different purpose (i.e. reusing), the usability perspectives for both purposes may differ or clash. Ontology reusability will be restricted depending on how different the usability perspectives are. We shall investigate this issue in section 2.2 since it is related to what we call ontology application-independence, or reusability verses usability.

Another important reusability concern is *the difficulty of identifying and isolating the reusable components*; i.e. allowing the general-purpose parts of an ontology to be reused instead of reusing the whole ontology. An ontology - in the common practice of ontology engineering - is being

The repeated use of the former ontology gives indication that it is widely accepted and there is a consensus about it, and it has been adequately tested and improved.

[11] See Appendix D for the definition of "domain level".

represented as one module. Internal couplings in knowledge structure (e.g. relationships between concepts, concept definitions, etc.) make it difficult for the general-purpose parts to be isolated and reused. For example, suppose one has a previously constructed a bookstore ontology that describes books, orders, shipping methods, payment methods, etc. It should be easy when building a new car-rental ontology to reuse for example, the payment aspects, since both Bookstore and Car-Rental ontologies share parts of a same axiomatization about payment methods. Ontology representation frameworks and languages should support modeling primitives that allow the representation of ontologies in a modular manner so that one can easily (de)compose modules.

Consequently, we believe that *the capability of ontology reuse strongly depends on the design and engineering of the ontology representation model.*

2.1.3 Conclusion

In this section we have defined what ontology reusability means, discussed the significance of ontology reusability as a fundamental requirement in ontology engineering; and clarified the main foundational challenges that restrict ontology reusability.

Based on the reusability challenges stated above, we derive the following ontology engineering requirements:

- Ontologies should be engineered in a way that allows the isolation and identification of the reusable parts of an ontology.
- The influence of usability perspectives on ontology axioms should not be emphasized during the ontology development phases[12].

In the next section, we proceed to discuss another *related* ontology engineering challenge.

[12] This requirement will be revisited and extended in the next section, we shall discuss the influence of usability perspectives in more detail.

2.2 Ontology application-independence

In this section, we discuss another *fundamental* ontology engineering challenge. We examine to what extent one can build an ontology independently of application requirements. Then, we discuss ontology reusability verses ontology usability before presenting the work done by other researchers in relation to this challenge. To end, we draw some important requirements for ontology engineering.

Ontologies are supposed to capture knowledge at the domain level independently of application requirements [G97] [GB99] [CJB99]. This is in fact, *the main and most fundamental asset of an ontology*. The greater the extent to which an ontology is independent of application requirements, the greater its reusability, and hence, the ease at which a consensus can be reached about it. Guarino argued in [G97] that:

> *"Reusability across multiple tasks or methods should be systematically pursued even when modeling knowledge related to a single task or method: the more this reusability is pursued, the closer we get to the intrinsic, task-independent aspects of a given piece of reality (at least, in the commonsense perception of a human agent)."*

Ontology application-independence is not limited to the independence of *implementation* requirements - it should also be considered at the *conceptual level.* For example, notice that application-independence is the main disparity between an ontology and a conceptual *data* schema (e.g. EER, ORM, UML, etc.) although both capture knowledge at the conceptual level [JDM03]. Unlike ontologies, when building a conceptual data schema, the modeling decisions depend on the specific needs and tasks that are planned to be performed within a certain enterprise, i.e. for "in-house" usage.

The problem is that when building an ontology, there will always be intended or expected usability requirements -"at hand"- which influence the independency level of ontology axioms. In the problem-solving research community, this is called the *interaction problem*. Bylander and Chandrasekaran argue that:

> *"Representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem."* [BC88]

The main challenge of usability influence is that different usability perspectives (i.e. different purposes of *what an ontology is made for* and *how it will be used*) lead to different - and sometimes conflicting - axiomatizations although these axiomatizations might agree at the domain level.

2.2.1 Example

The following example illustrates the influence of some usability perspectives when modeling Bibliography ontologies.

We present two ontologies within the same Bibliography domain: ontology A in fig. 2.1 and ontology B in Fig. 2.2. Suppose that both ontologies are built separately; ontology A is built and used within a community of bookstores, and ontology B is built and used within a community of libraries[13].

We will show that although both ontologies intuitively agree at the domain level, they differ formally because of the differences in their communities' usability perspectives. To this end, we argue that building ontologies under the strong influence of usability perspectives leads to more application-dependent, and thus less reusable ontologies.

[13] Notice that the goal of this example is neither to discuss the Bibliography domain itself, nor to present adequate an ontology - we use it only for illustration purposes.

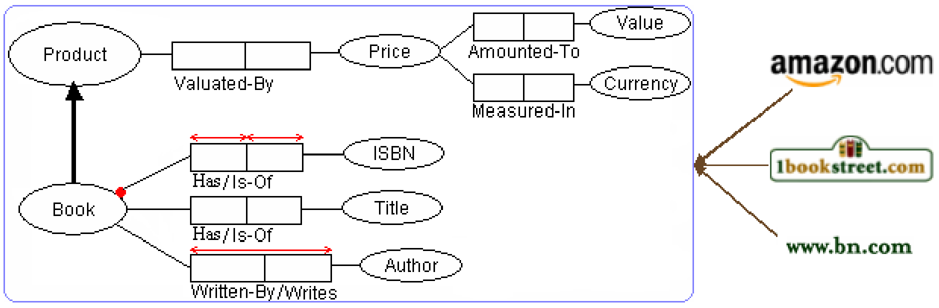


Fig. 2.1. Ontology A.

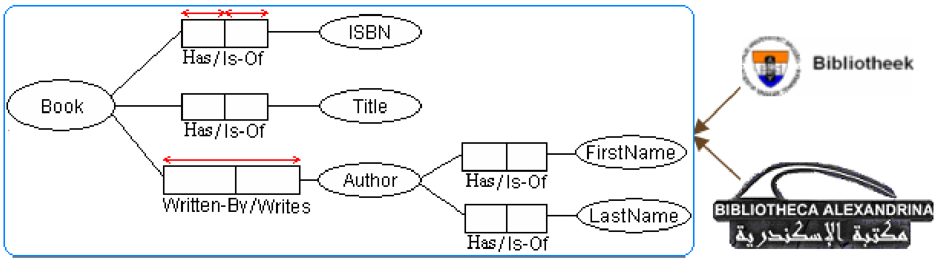


Fig. 2.2. Ontology B.

In the following, we examine the influence of usability perspectives on the modeling decisions of both *conceptual relations*[14] and *ontology rules*,[15] respectively.

On modeling conceptual relations. The concept 'Author' in ontology B is attributed with the 'First Name' and the 'Last Name' concepts. Such details (i.e. *granularity*) are not *relevant* to bookstore applications; they are not specified in ontology A. Similarly, unlike ontology A, the pricing relations {Valuated-By(Book, Price), Amounted-To(Price, Value), Measured-In(Price, Currency)} are not *relevant* for library applications, so they are not specified in ontology B.

From such differences, one can see that deciding the *granularity level* and the *scope boundaries* depend on the relevance to the intended (or expected) usability. Although such differences do not necessarily

[14] See appendix D for the definition of "conceptual relation".
[15] See appendix D for the definition of "ontology rule".

-D

constitute a disagreement between both axiomatizations, they hamper the reusability of both ontologies. In order to reuse such ontologies, the reusing applications need to make some adaptations, *viz. introducing* the incomplete knowledge and *dismissing* the "useless" knowledge that normally distracts and scales down the reasoning/computational processes.

On modeling ontology rules. Notice that both ontologies in the example above do not agree on the notion of what is a "Book". Although both ontologies agree that the ISBN is a unique property for the concept book (see the uniqueness rules[16]), they disagree whether this property is mandatory for each instance of a book. Unlike ontology B, ontology A axiomatizes that each instance of a book *must* have an ISBN value (see the mandatory rule[17]). This rule implies for example that "PhD Theses" or "Manuals", etc. would not be considered instances of books in ontology A because they do not have an ISBN, while they would be under ontology B.

One can see from this example that modeling the ISBN as mandatory property for all instances of the concept book is naturally affected by bookstores' business perspective. Obviously, bookstores communicate only the books "that can be sold" and thus "commercially" should have ISBN, rather than perusing the notion of book at the domain level. Nevertheless, at the domain level, both bookstore and library applications intuitively share the same concept of what is really a book. For example, suppose that one assigns an ISBN for an instance of a "PhD Thesis". This instance can then be considered as a book for bookstores. If however, the ISBN is removed for an instance of a book, then this instance will no

[16] The uniqueness rule in ORM is equivalent to 0:1 cardinality restriction. (notation: '⟷'), it can be verbalized as "each book must have at most one ISBN".

[17] The mandatory rule in ORM is equivalent to 1-m cardinality restriction. (notation: '●'), it can be verbalized as "each book must have at least one ISBN".

longer be a book, even though it still refers to the same real life object and is still being referred to and used as a book.

Accordingly, as ontology rules are supposed to formally specify/constrain the permitted models[18] that can necessarily hold for a given domain [F02], determining such rules, in practice is dominated by "what is permitted and what is not" for the *intended* or *expected* usability.

Furthermore, besides the modeling decisions of ontology rules, the determination of the number and the type of these rules (the reasoning scenario) are also influenced by usability perspectives. For example, a light-weight axiomatization (e.g. with a minimum number of rules or formalities) might be sufficient if the ontology is to be accessed and used by people (i.e. not computers). Depending on the application scenario, other *types* of ontology rules (i.e. modeling primitives/constructs) might be preferred, over the ORM set of rules (which are easier to reason for database and XML based applications).

At this point, we conclude that even application-types might intuitively agree on the same semantics at the domain level, but the usability influence on axiomatizing this semantics may lead to different (or even conflicting) axiomatizations. An axiomatization might be more relevant for some applications than others, due to the difference of their usability perspectives. This issue presents an important challenge to the nature and the foundation of ontology engineering.

2.2.2 Related work

Guarino and his co-authors have argued (in e.g. [G98a][G97]) that in order to capture knowledge at the domain level, the notion of what is an ontology should be more precisely defined. Gruber's commonly used definition, [G95], of an ontology is of "an explicit specification of a conceptualization", referring to an *extensional* ("Tarski-like") notion of a

[18] Also called "ontology models" as in [G95].

conceptualization as found e.g. in [GN87]. Guarino and his collaborators point out that this definition *per se* does not adequately fit the purposes of an ontology. They argue, in our opinion correctly, that a conceptualization *should not be extensional b*ecause a conceptualization benefits from invariance under changes that occur at the instance level and from transitions between different "states of affairs"[19] in a domain. They propose a conceptualization as *an intensional* semantic structure i.e. abstracting from the instance level, which encodes *implicit rules* constraining the structure of a piece of reality[20]. Therefore, "an ontology only indirectly accounts for a conceptualization". In other words, an ontology becomes a logical theory which possesses a conceptualization as an explicit, partial model. Furthermore, they have proposed the OntoClean methodology for evaluating ontological decisions [GW02]. The methodology consists of a set of formal notions that are drawn from Analytical Philosophy and called metaproperties. Such metaproperties include rigidity, essence, identity, unity, and dependence. The idea of these notions is to focus on the *intrinsic properties* of the concepts, which are *application-independent.*

Following Guarino et al's ontology definition and their associated OntoClean methodology, one can see in the previous example that the two axiomatizations should not be seen as different ontologies since they only differ on their description of extensions i.e. states of affairs. Both axiomatizations implicitly share the same intensional semantic structure or conceptualization. Furthermore, the ISBN is an extrinsic property (i.e. not intrinsic)[21] since it is not rigid[22] for all instances of the concept book.

[19] See Appendix D for the definition of "state of affairs".

[20] See e.g. the definition of "extensional verses intensional semantics" in appendix D.

[21] To understand the difference between intrinsic and extrinsic properties, the following is a quotation taken from [GW00]: "*An intrinsic property is typically something inherent to an individual, not dependent on other individuals, such as having a heart or having a fingerprint. Extrinsic properties are not inherent, and they have a relational nature, like "being a friend of John". Among these, there are some that are typically assigned by*

-D

Therefore, it cannot be used to specify the intended meaning of a book at the domain level.

An important problem of the OntoClean methodology, in our opinion, is its applicability. It relies on deep philosophical notions that (1) in practice are not easy or intuitive to utilize - at least for "nonintellectual" domain experts; and (2) it only focuses on the intrinsic properties of concepts and such properties are often difficult to articulate. For example, how to formally and explicitly articulate the identity criteria of a book (or person, brain, table, conference, love, etc.). Guarino and Welty state in [WG01]: "*We may claim as part of our analysis that people are uniquely identified by their brain, but this information would not appear in the final system we are designing*". In short, it would seems that OntoClean can be applied mainly by highly trained intellectuals for domain analysis and ontological checks[23].

2.2.3 Ontology usability is also important

There is another factor that should not be ignored, especially with regards to the philosophically inspired research on ontologies (or the so-called "philosophical ontology" as in [S03a]). In keeping with current views in the field of information technology, ontologies are to be shared and used collaboratively in software applications. This gives even more weight to the importance of *ontology usability*.

2.2.4 Conclusion

The closer an axiomatization is to certain application perspectives, the more usable it will be. In contrast, the more an axiomatization is independent of application perspectives, the more reusable it will be. In

external agents or agencies, such as having a specific social security number, having a specific customer i.d., even having a specific name."

[22] "A property is rigid if it is essential to all its possible instances; an instance of a rigid property cannot stop being an instance of that property in a different world" [WG03].

[23] See [GGO02] for a successful application of OntoClean on cleaning up WordNet[M95].

other words, *there is a tradeoff between ontology usability and ontology reusability*.

From a *methodological viewpoint,* if a methodology emphasizes usability perspectives or evaluates ontologies based on how they fulfill specific application requirements, the resulting ontology will be similar to a conceptual data schema (or a classical knowledge base) containing application specific and thus, less reusable knowledge. Likewise, if a methodology emphasizes the independency of the knowledge, the resulting ontology in general will be less *usable*, since it has no intended use by ignoring application perspectives.

Based on the above, we propose the following ontology engineering requirement:

- The influence of usability perspectives on ontology axioms should be well articulated, pursuing both reusability and usability.

To fulfill this requirement, in Chapter 3 we will propose the ontology double articulation principle. Concisely, an ontology is double-articulated into domain axiomatization and its application axiomatizations. While a domain axiomatization is concerned with capturing the intended meaning of domain vocabularies (which is supposed to be reusable), application axiomatizations are mainly concerned with the usability of these vocabularies.

We are now ready to analyze the third ontology engineering challenge: ontology evolution.

2.3 Ontology evolution

The continuous growth and intensive maintenance of ontologies are serious concerns in the ontology development life cycle. Ontologies evolve over time [KKOF02], due to conceptual changes, epistemological[24] changes, scope extensions, mistake corrections and quality enhancements, etc. Furthermore, ontologies evolve in a distributed environment through interactions by different people over different locations [BHGSS03].

Current research on ontology evolution focuses mainly on treating the implication of changes on the applications that are committing to a "changed" ontology, more than dealing with the evolution process itself. Change to an ontology has operational consequences for running applications – for example, consider the implications of changes on a database schema [VH91]. Various mechanisms have been proposed to tackle the impact of changes by separating the changes into new versions or layers, see e.g. [Hj01] [KKOF02] [MMS03].

Not only the implications of evolution, but also the evolution process "itself" becomes more complex in case of large-scale and distributed ontologies. In this thesis, we focus only on clarifying and tackling foundational (i.e. not technical) challenges in ontology evaluation.

2.3.1 The complexity of change

Before modifying or extending an ontology, one needs to carefully understand the *intended meaning* of all existing concepts and axioms. In case of large-scale ontologies, this process becomes more complex because (1) of the internal couplings among axioms and the large number of them; (2) the large-scale ontologies are usually built by different people and capture knowledge across domains and subjects.

[24] See appendix D for the definition.

As ontology axioms only indirectly account for a conceptualization [G98a], a large part of the intended meaning of the ontology concepts will remain *implicit* between ontology developers. It will be difficult for *different* ontology developers, especially those with different backgrounds working in different time periods to know what was originally intended, or what the modeling decisions and choices were. To a large extent, the literal interpretations of the concepts labels (i.e. terms) will be considered rather than what was originally intended, especially in case of a light-weight ontology axiomatization.

Accordingly, in order to achieve efficient maintenance, *critical assumptions that are important because they make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology. Such an attachment - even if added informally - would facilitate both users' and developers' commonsense perception of the subject matter.* It is important, not only for future maintenance but also advised for the collaborative and distributed development of ontologies. To fulfill this engineering requirement, we shall introduce the notion of gloss to ontology engineering in chapter 3. A gloss is supposed to render informally the factual knowledge that is critical to understanding a concept, but that is unreasonable, irrelevant, or very difficult to formalize and articulate explicitly.

2.3.2 Distributed evolution

Ontology development and maintenance is not a single-person effort. Adequate ontologies are normally built, reviewed, and maintained by several types of knowledge experts [SK03]. For example, our experience in building a “Customer Complain Ontology”, reported in chapter 7, shows that some parts of the ontology - specifically those that capture knowledge about customer regulations - should be built and evaluated by lawyers. The classification of complaint problems and resolutions should be performed by market and ADR[25] experts. The whole ontology needs to be reviewed by CRM[26] application experts and other such professionals.

Engineering such collaborations is a challenge, especially in the case of large-scale and multi-domain ontologies. First, the development and maintenance processes need to be *divided* and *distributed* among the contributors according to their expertise; second, the contributions of the experts need to be integrated and this is not an easy task.

Several software environments have been proposed to enable the distributed development of ontologies, such as [SKKM03], [MMS03], and [TTN97]. We believe that instead of (or complementary to) developing such ad hoc tactics for tackling this issue, *the ontology representation model itself should be capable of distributed development and smooth evolution*[27]. As an analogy, compare the capability of distributing the development of a program built in Pascal with a program built in JAVA i.e. procedural verses object-oriented distributed software development.

2.3.3 Alternative axiomatizations

Alternative axiomatizations are different formalizations of the same subject-matter. They reflect different usability perspectives. As we have discussed in section 2.2, an axiomatization might be more relevant or usable for one application than another. In many cases, the irrelevance might only apply to certain portions and not the whole axiomatization. For example, the creators of different applications may prefer to alter the axiomatization of the notion of 'address' within an ontology depending on how addresses are structured in their country of service.

The main advantages of allowing easy interchange of ontology parts (i.e. replacing parts), in general, are:

[25] ADR stands for Alternative Dispute Resolution.
[26] CRM stands for Customer Relationship Management.
[27] In chapter 4 and 5, we shall discuss and illustrate how the double-articulation and the modularization engineering principles aim to fulfill this requirement.

1. Enabling ontology users and maintainers to interchange ontology parts with others that are more relevant, reliable, accurate, etc.
2. Enabling "Natural" ontology evolution: successful axiomatizations in certain domains will likely become popular and evolve into the trusted de facto semantics.

Still, the way an ontology is represented and engineered currently does not allow for an easy interchange of it parts as it is being built and used as one component. *Alternating the axiomatization of ontology parts demands that the ontology be represented and engineered as a configurable set of modules*; rather than as one large and complexly interrelated component[28].

2.3.4 Conclusion

In this section, we have presented important engineering challenges in ontology evolution: complexity of change, distributed development, and alternating axiomatizations.

Based on the above challenges, we draw the following ontology engineering requirements:

- Critical assumptions that make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology, even if informally, to facilitate both users' and developers' commonsense perception of the subject matter.
- The ontology representation model should be capable of distributed and collaborative development.
- Ontologies should be engineered in a way that enables smooth and efficient evolution.
- Ontologies should be engineered in a way that allows for easy replacement of ontology parts.

[28] We shall discuss ontology modularization in chapter 5, and illustrate (de/)compose of ontological modules.

2.4 Summary

In this chapter, we have focused on clarifying several foundational challenges in ontology engineering: ontology reusability, ontology application-independence, and ontology evolution[29]. Based on these challenges, we summarize the main ontology engineering requirements in the table below:

No.	Requirement
R1.	Ontologies should be engineered in a way that allows the isolation and identification of the reusable parts of the ontology.
R2.	The influence of usability perspectives on ontology axioms should be well articulated, in pursuit of both reusability and usability.
R3.	Critical assumptions that make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology, even if informally, to facilitate both users' and developers' commonsense perception of the subject matter.
R4.	The ontology representation model should be capable of distributed and collaborative development.
R5.	Ontologies should be engineered in a way that enables smooth and efficient evolution.

[29] We are aware of other foundational challenges in ontology engineering that are not discussed due to the limited focus of our research. Such challenges include that of ontology multilingualism and ontology integration. We have developed modest methodological guidelines for developing multilingual lexicalization of ontologies. These guidelines are presented briefly in chapter 7, as part of our case study. Furthermore, [K04] [VDZ04] show some advantages and applications of our methodological principles in ontology integration.

R6.	Ontologies should be engineered in a way that allows for the easy replacement of ontology parts.

Table 2.1. Ontology Engineering Requirements.

As outlined earlier, this thesis is structured in three parts. We specify the problem, propose a solution, and show an implementation of this solution. In this chapter, we have specified the ontology engineering challenges and derived some engineering requirements. Fulfilling these requirements is the goal of our methodological principles and we present these in the next part of this thesis.

-D

Part II

Methodological principles

The term 'methodology' means:

"New Latin methodologia, from Latin methodus + -logia –logy 1)a body of methods , rules, and postulates employed by a discipline **:** *a particular procedure or set of procedures. 2) the analysis of the principles or procedures of inquiry in a particular field."*

-(Merriam-Webster Online Dictionary)

In this part, we introduce our methodological principles for ontology engineering, namely the ontology double articulation principle (chapter 3) and the ontology modularization principle (chapter 4).

-D

Chapter 3

Ontology Double Articulation

"Syntax is merely a necessary device by which we attach semantics to the representation, and it makes little sense to claim that a representation formalism is semantically more powerful merely because it has more syntactical constructs …"

-(Robert Meersman, [M95],)

This chapter presents the first engineering principle: ontology double articulation (a domain axiomatization and its application axiomatizations). Section 3.1 quickly introduces the double articulation principle. In section 3.2, we present and discuss the general properties of domain axiomatization. Section 3.3 introduces the notion of ontology base for capturing domain axiomatizations. In section 3.4 we discuss the nature of application axiomatizations, and introduce the notion of application ontological commitments. Finally, section 3.5 summarizes the main advantages that can be gained and the engineering requirements that can be fulfilled by the double articulation principle.

3.1 Introduction[30]

In this section, we schematically introduce and illustrate the principle and its general idea. Further details follow.

The goal of the ontology double articulation principle[31], mainly, is to fulfill the R2 engineering requirement: The influence of usability perspectives on ontology axioms should be well articulated, in pursuit of both reusability and usability.

As we have noted earlier, our research on ontology double articulation is based on the research conducted by Meersman in [M99a] [M99b] within the DOGMA project. In this chapter we introduce fundamental changes and extensions.

The term "double articulation", in this thesis, simply means *expressing knowledge in a twofold axiomatization*. See section 3.2 for the formal definition and details. The term "articulation" in WordNet means: "Expressing in coherent verbal form", "The shape or manner in which things come together and a connection is made", etc. In the semiotics and linguistics literature, the term "double articulation" has been introduced by [N90][M55][32] (which has a different meaning and usage than ours) to refer to the distinction between lexical and functional unites of language or between content and expression.

3.1.1 Overview of the double articulation principle

The ontology double articulation principle, in nutshell, is that: *an ontology is doubly articulated into: domain axiomatization and application axiomatizations. While a domain axiomatization is mainly concerned with*

[30] Later this chapter was revised and extended (see [JM08])
[31] In this chapter, we, sometimes, refer to this principle as "the principle" or "this principle".
[32] We are grateful to Dr. Peter Spyns for drawing our attention to this analogy and introduction of this term.

characterizing the "intended meanings" of domain vocabulary (typically shared and public), an application axiomatization (typically local) is mainly concerned with the usability of these vocabularies. The double articulation implies that all concepts and relationships introduced in an application axiomatization are predefined in its domain axiomatization. Multiple application axiomatizations (e.g. that reflect different usability perspectives, and that are more usable) share and reuse the same intended meanings in a domain axiomatization.

To translate this principle into software architecture, see DOGMA[33], we adopt (and extend) the notion of *ontology base* [M99a] for capturing domain axiomatizations; and we introduce the notion of application axiomatization, by which particular applications commit to an ontology base. An ontology therefore can be seen as an ontology base and a *layer* of ontological commitments, i.e. a *domain axiomatization and its application axiomatizations*, see fig. 3.1.

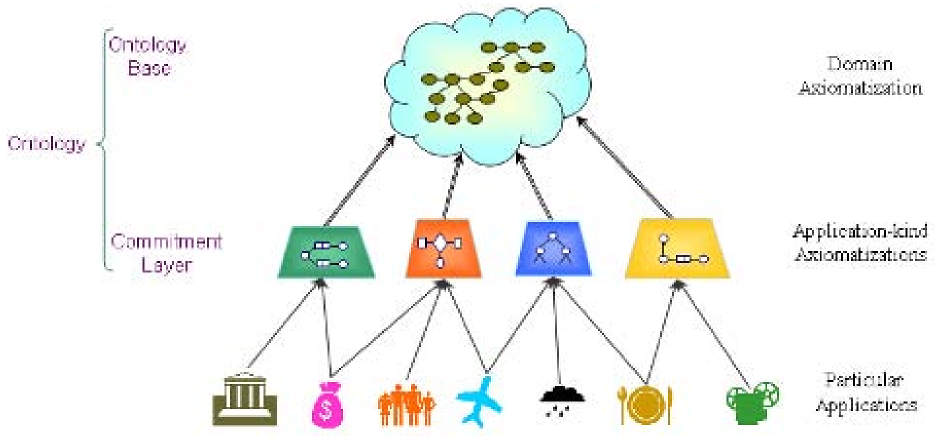


Fig. 3.1. Ontology Double Articulation.

The ontology base is intended to capture domain axiomatizations. It basically consists of a set of binary conceptual relations [M99a]. The lexical rendering of a binary conceptual relation is called *lexon*. A lexon is

[33] See http://www.starlab.vub.ac.be/research/dogma.htm (March 2005)

-D

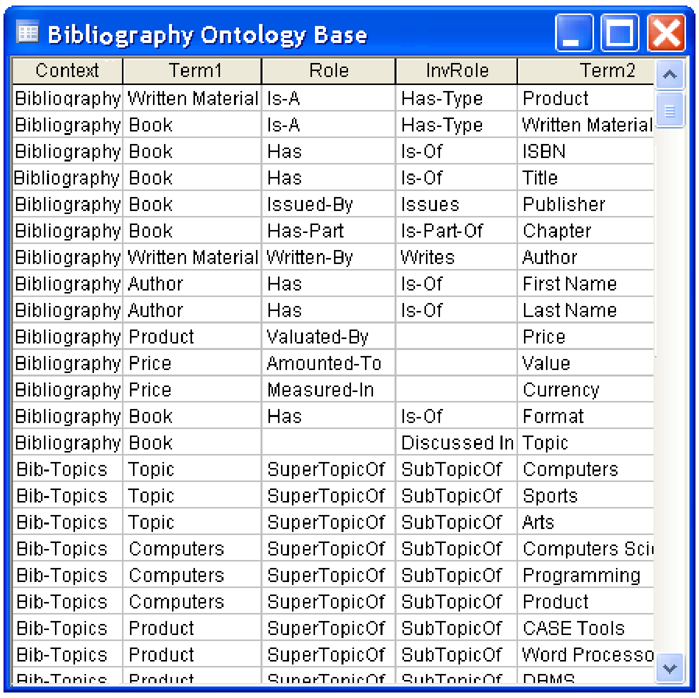
described as a tuple of the form <γ: Term_1, Role, InvRole, Term_2>, where Term_1 and Term_2 are linguistic terms. γ is a context identifier, used to *bound* the interpretation of a linguistic term into a concept. For each context γ and term T, the pair (γ, T) is assumed to refer to a *concept*. Role and InvRole are lexicalizations of the pair roles of a binary relationship R, e.g. HasType/IsTypeOf.

The commitment layer consists of a set of application axiomatizations. Particular applications commit to the ontology base through an application axiomatization. Such a commitment is called application *ontological commitment*[34]. Each application axiomatization consists of: (1) a set of lexons from an ontology base; (2) a set of rules to constrain the usability of these lexons.

3.1.2 Example

In this example, we re*visit* the bibliographic example that we presented in section 2.2. Fig. 3.2 shows a Bibliography ontology base.

[34] We sometimes use the notion of "application ontological commitment" and the notion "application axiomatization" interchangbly in this thesis. It is also worth to note that the notion of "ontological commitment" as found in [GG95] generally refers to a "conceptualization", literally, it is defined as "a partial semantic account of the intended conceptualization of a logical theory."

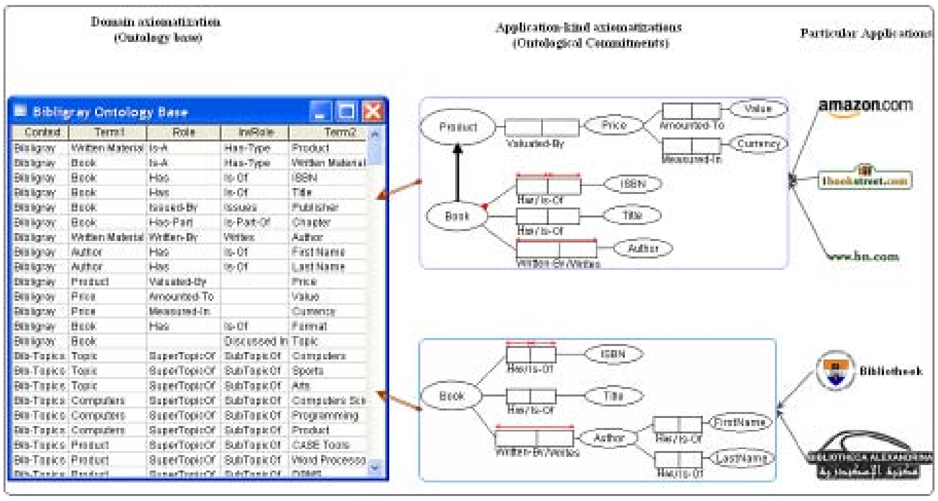


Context	Term1	Role	InvRole	Term2
Bibliography	Written Material	Is-A	Has-Type	Product
Bibliography	Book	Is-A	Has-Type	Written Material
Bibliography	Book	Has	Is-Of	ISBN
Bibliography	Book	Has	Is-Of	Title
Bibliography	Book	Issued-By	Issues	Publisher
Bibliography	Book	Has-Part	Is-Part-Of	Chapter
Bibliography	Written Material	Written-By	Writes	Author
Bibliography	Author	Has	Is-Of	First Name
Bibliography	Author	Has	Is-Of	Last Name
Bibliography	Product	Valuated-By		Price
Bibliography	Price	Amounted-To		Value
Bibliography	Price	Measured-In		Currency
Bibliography	Book	Has	Is-Of	Format
Bibliography	Book		Discussed In	Topic
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Computers
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Sports
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Arts
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Computers Sci
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Programming
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Product
Bib-Topics	Product	SuperTopicOf	SubTopicOf	CASE Tools
Bib-Topics	Product	SuperTopicOf	SubTopicOf	Word Processo
Bib-Topics	Product	SuperTopicOf	SubTopicOf	DBMS

Fig. 3.2. A bibliography ontology base.

The illustrations in figures 2.1 and 2.2 are seen as two application axiomatizations (Bookstore and Library axiomatizations) by which particular applications make a commitment to and share the same Bibliography ontology base (see figure 3.3). *Notice that all conceptual relations in both application axiomatizations correspond to (or are derived from) lexons in the Bibliography ontology base. In this way, different application axiomatizations share and reuse the same intended meaning of domain concepts.*

-D



Context	Term1	Role	InvRole	Term2
Bibligray	Written Material	Is-A	Has-Type	Product
Bibligray	Book	Is-A	Has-Type	Written Material
Bibligray	Book	Has	Is-Of	ISBN
Bibligray	Book	Has	Is-Of	Title
Bibligray	Book	Issued-By	Issues	Publisher
Bibligray	Book	Has-Part	Is-Part-Of	Chapter
Bibligray	Written Material	Written-By	Writes	Author
Bibligray	Author	Has	Is-Of	First Name
Bibligray	Author	Has	Is-Of	Last Name
Bibligray	Product	Valuated-By		Price
Bibligray	Price	Amounted-To		Value
Bibligray	Price	Measured-In		Currency
Bibligray	Book	Has	Is-Of	Format
Bibligray	Book		Discussed In	Topic
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Computers
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Sports
Bib-Topics	Topic	SuperTopicOf	SubTopicOf	Arts
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Computers Sci
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Programming
Bib-Topics	Computers	SuperTopicOf	SubTopicOf	Product
Bib-Topics	Product	SuperTopicOf	SubTopicOf	CASE Tools
Bib-Topics	Product	SuperTopicOf	SubTopicOf	Word Processo
Bib-Topics	Product	SuperTopicOf	SubTopicOf	DBMS

Fig. 3.3. Particular applications committing to an ontology base through their application axiomatizations.

3.2 Domain Axiomatization

In the previous section, we have briefly introduced the ontology double articulation principle. In this section, we discuss[35] the general properties of domain axiomatization[36], *viz.* the nature and the level of details that are appropriate to characterize *domain concepts*.

As we have discussed in section 2.2, decreasing the influence of usability perspectives is a principal engineering requirement when axiomatizing *domain concepts*. To capture knowledge at the domain level, one should focus on characterizing the *intended meaning* of domain vocabularies (i.e. domain concepts), rather than on how and why these concepts will be used. A domain axiomatization becomes an axiomatic theory that only includes the axioms that account for (i.e. characterize) the intended meaning of the domain vocabularies.

[35] Our *style* of discussion in this section is inspired by the style used by Nicola Guarino and Barry Smith to discuss what is an ontology, conceptualization, etc.
[36] These properties are summarized at the end of this section.

-D

This motivates us to understand the relationship between a domain vocabulary and the specification of its intended meaning in a logical theory.

In general, it is not possible to build a logical theory to specify *the complete and exact intended meaning* of a domain vocabulary[37]. Usually, the level of detail that is appropriate to explicitly capture and represent it is subject to what is reasonable and plausible for domain applications. Other details will have to remain implicit assumptions. These assumptions are usually denoted in linguistic terms that we use to lexicalize concepts, and this implicit character follows from our interpretation of these linguistic terms.

On the relationship between concepts and their linguistic terms Avicenna (980-1037 AC) [Q91] argued that:

> *"There is a strong relationship/dependence between concepts and their linguistic terms, change on linguistic aspects may affect the intended meaning… Therefore logicians should consider linguistic aspects 'as they are'. …"*[38].

Indeed, the linguistic terms that we usually use to name symbols in a logical theory convey some important assumptions, which are part of the conceptualization that underlie the logical theory. We believe that these assumptions should not be excluded or ignored (at least by definition) as indeed *they are part of our conceptualization*.

Hence, we share Guarino and Giaretta's viewpoint [GG95], that an ontology (as explicit domain axiomatization) only *approximates* its underlying conceptualization; and that a domain axiomatization should be

[37] This is because of the large number of axioms and details that need to be intensively captured and investigated, such detailed axiomatizations are difficult -for both humans and machines- to compute and reason on, and might holds "trivial" assumptions.
[38] This is an approximated translation from Arabic to English.

interpreted intensionally, referring to the intensional notion of a conceptualization.

Gruber [G95] defined an ontology as an explicit specification of a conceptualization, referring to the *extensional* ("Tarski-like") notion of a conceptualization as found in [GN87]. Guarino and Giaretta pointed out that this definition *per se* does not adequately fit the purposes of an ontology. They argued that according to Gruber's definition, the re-arrangement of domain objects (i.e. different state of affairs) corresponds to *different* conceptualizations. Guarino and Giaretta argue that a conceptualization benefits from invariance under changes that occur at the instance level by transitions between merely different "states of affairs" in a domain, and thus *should not be extensional*. Instead, they propose a conceptualization as *an intensional semantic structure* (i.e. abstracting from the instance level), *which encodes implicit rules* constraining the structure of a piece of reality[39]. Indeed, this definition allows for the focus on the meaning of domain vocabularies (by capturing their intuitions) independently of a state of affairs. See [G98a] for the details and formalisms.

3.2.1 Definition (double articulation, intended models, legal models)

Given a concept **C** as *a set of rules (i.e. axioms) in our mind about a certain thing in reality*, the set **I** of "all possible" instances that comply with these rules are called the *intended models* of the concept **C**. According to the ontology double articulation principle, such concepts are captured at the domain axiomatization level. An application A_i that is interested in a subset $\mathbf{I}_{Ai}$ of the set **I** **(**according to its usability perspectives**)**, is supposed to provide some rules to specialize **I**. In other words, every instance in $\mathbf{I}_{Ai}$ must also be an instance in **I**:

$$\mathbf{I}_{Ai} \subseteq \mathbf{I}$$

[39] See the definition of "Extensional verses Intensional semantics" in appendix D.

We call the subset $\mathbf{I}_{Ai}$: the *legal models* (or extensions) of the application's concept $\mathbf{C}_{Ai}$. Such application rules are captured at the application axiomatization level.

Both domain and application axiomatizations can be seen (or expressed) as sentences in first order logic.

As we have illustrated in the previous section, bookstore applications that are interested *only* in the instances of the concept 'book' (that can be sold) need to declare the Mandatory rule that each instance of book must have an ISBN value.

In Fig. 3.4 we show three kinds of applications specializing a domain concept.

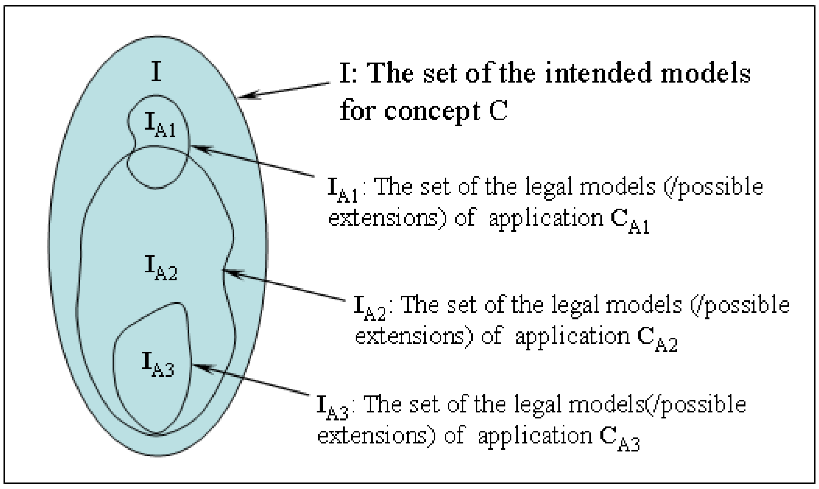

I
I_{A1}
I_{A2}
I_{A3}
I: The set of the intended models for concept C
I_{A1}: The set of the legal models (/possible extensions) of application C_{A1}
I_{A2}: The set of the legal models (/possible extensions) of application C_{A2}
I_{A3}: The set of the legal models(/possible extensions) of application C_{A3}

Fig. 3.4. An example of three different applications specializing a domain concept.

The differences between the legal models of these application-types illustrate their different usability perspectives:

- The intersection between the legal models of $\mathbf{C}_{A2}$ and the legal models $\mathbf{C}_{A3}$ shows that I_{A3} is a subset of I_{A2}. An example of this case could be the difference between notions of 'book' in the axiomatization of bookstores and libraries: all legal instances of the bookstores' notion are legal instances for the libraries, but not vice versa. For libraries, the instances of e.g. 'Manual' or 'Master Thesis' can be instances of a 'book'; however, they cannot be instances of 'book' for bookstores, unless they are published with an 'ISBN'.

- The difference between $\mathbf{I}_{A1}$ and $\mathbf{I}_{A3}$ shows an extreme case: two types of applications sharing the same concept **C** while their legal models are completely disjoint according to their usability perspectives. An example of this case could be the difference between notions of 'book' in the axiomatization of bookstores' and museums': Museums are interested in exhibiting and exchanging instances of old 'books', while bookstores are not interested in such 'books', unless for example, they are re-edited and published in a modern style.

One may wonder how *domain concepts* can be agreed upon because of the difficulty in gaining an objective insight into the nuances of another person's thoughts. Many researchers admit that a conceptualization reflects a particular viewpoint and that it is entirely possible that every person has his own concepts. For example, Bench-Capon and Malcolm argued in [BM99] that conceptualizations are likely to be influenced by personal tastes and may reflect fundamental disagreements. In our opinion, herein lies the importance of linguistic terms.

3.2.2 Importance of linguistic terms in ontology engineering

Linguistic resources (such as lexicons, dictionaries, and glossaries.) can be used as consensus references to *root* ontology concepts. In other words, ontology concepts and axioms can be investigated using such linguistic resources and it can be determined whether a concept is influenced by personal tastes or usability perspectives. We explain this idea further in the following paragraphs:

The importance of using linguistic resources in this way lies in the fact that a linguistic resource renders the intended meaning of a linguistic term as it is commonly agreed among the community of its language. The set of concepts that a language lexicalizes through its set of word-forms is generally an agreed conceptualization[40][T00]. For example, when we use the English word 'book', we actually refer to the set of implicit rules that are common to English-speaking people for distinguishing 'books' from other objects. Such implicit rules (i.e. concepts) are learned and agreed from the repeated use of word-forms and their referents. Usually, lexicographers and lexicon developers investigate the repeated use of a word-form (e.g. based on a comprehensive corpus) to determine its underlying concept(s) [BDVHP00] [RFOGP99].

Given the definition of the term 'book' found in WordNet (a written work or composition that has been published, printed on pages bound together), one can judge, for example, that an ISBN is not really a necessary property for every instance of a book (see our discussion in section 2). Notice that such judgments cannot be based on the literal interpretation of the term definition, but should be based on the *intuition* that such short definitions provide. For more precision, one may use several linguistic resources to investigate and root ontology concepts.

In short, a way of preventing ontology builders from imposing their personal viewpoints and usability perspectives at the conceptual level is, by investigating and rooting the ontology concepts at the level of a human language conceptualization. This involves making a distinction between a personal viewpoint and. a community viewpoint. Notice that by doing this, we are (indirectly) investigating and rooting our ontology concepts at the domain level, because the conceptualization of a language emerges from the repeated use of linguistic terms and their referents in real life domains.

[40] Thus, we may view a lexicon of a language as an informal ontology for its community.

Taking a step further in this regard, we will discuss and illustrate the incorporation of existing linguistic resources into the ontology engineering process in section 3.5 and 6.2.2. We shall show how to link the vocabulary used in an ontology with term-definitions found in linguistic resources. In section 3.3.6 we shall introduce the notion of gloss to capture such definitions, and to define new concepts that may not exist in linguistic resources.

3.2.3 On representing domain axiomatizations

In this section, we discuss some choices that we think are relevant for *representing* domain axiomatizations.

A domain axiomatization merely cannot be a list of linguistic terms, and their intended meanings cannot be completely implicit. The intended meaning of linguistic terms should be axiomatized and represented by means of a formal language.

From a methodological viewpoint, such a formal language should be content-oriented rather than syntax-oriented. This language should serve as a theoretical tool which guides ontology builders through its primitives, and restrict them to focus *only* on and represent the “kinds” of axioms that account for the intended meaning of domain vocabularies.

By analogy, the conceptual “data” modeling languages ORM and EER provide database designers a set of primitives with which they can be guided to build a normalized database schema. Indeed, ORM and EER can be seen as content-oriented languages, because they *restrict* the focus of database designers to the *integrity of data models*.

An example of the difference between conceptual data modeling primitives and the kind of primitives that account for the intended meaning of a vocabulary[41] is the difference between the “Rigid” and “Mandatory”. Something can be mandatory but not rigid, as in the case of

[41] i.e. conceptual data modeling vs. conceptual domain modeling.

‘ISBN’ which is not a rigid property for every instance of a ‘book’ but could be mandatory for some applications. In other words, to model something as a rigid property, it should be rigid in *all possible* applications, while what can be mandatory for an application might not be mandatory for another. See [GW00][JDM03][WSW99][GHW02] for more discussions on such issues.

Current research trends on ontology languages within the Semantic Web and the description logic communities are mainly concerned with improving *logical consistency* and inference services. Such services in our opinion are more suitable for building knowledge base applications or expert systems rather than axiomatizing “domain concepts”. Significant results within the description logic community have indeed been achieved in the development of expressive and decidable logics, such as ***DLR*** [CDLNR98], SHIQ [HST99], SHOQ [HS01], etc., yet less attention has been given to the quality of ontological *content*.

> *“…I was annoyed by the fact that knowledge representation research was more and more focusing on reasoning issues, while the core problems of getting the right representations were not receiving that much attention…”. (Nicola Guarino[42]).*

An example of a modeling primitive in the SHOQ description logic which in our opinion, should not be allowed in axiomatizing domain concepts since it does not account for meaning, is *datatypes* [P04]. Such a primitive belongs mainly to the symbolic level. In short, description logics (and their derivative languages such as DAML+OIL, or OWL) seem to play a useful role in specifying application (rather than domain) axiomatizations.

We shall return, in section 3.4 to the use of both conceptual data modeling languages and description logic based languages, for modeling and representing application axiomatizations.

[42] An interview with Nicola Guarino and Christopher Welty (9 June 2004): http://esi-topics.com/erf/2004/june04-ChristopherWelty html

-D

We observe two possible ways to capture formal domain axiomatizations: (1) as an arbitrary set of axioms, e.g. using description logic, or (2) through a knowledge representation model (e.g. a database). The first case is common within the Semantic Web and Artificial Intelligence communities; in this case ontology builders are responsible (i.e. unguided) to decide whether an axiom accounts for the intended meaning of a vocabulary. This way offers ontology builders more freedom and expressiveness, but the risk of encoding usability perspectives is still high. In the second case, ontology builders are restricted only to capturing and storing the kind of axioms that account for factual meaning; assuming that the representation model is well studied and designed to pursue such axioms. This way is less expressive than the first one, but it reduces the risk of mixing domain and application axioms. The second way offers scalability in accessing and retrieving axioms, which is usually a problematic issue in the first way. The second way is mostly used within the lexical semantics community, e.g. WordNet [MBFGM90], Termintography [KTT03]. Notice that both ways are (or should be) well formalized and map-able to first order logic, and thus can be seen as logical theories.

We have chosen the second way for our approach. As we will show in section 3.3, we have developed a data model for capturing domain axiomatizations called an *ontology base* [M99a][M99b].

3.2.4 Summary: properties of domain axiomatization

In this section, we summarize the basic properties of a domain axiomatization: it is (1) an axiomatized theory (2) that accounts for the intended meaning of domain vocabularies; (3) it is intended to be shared and used as a vocabulary space for application axiomatizations. It is supposed to be (4) interpreted intensionally, (5) and investigated and rooted at a human language conceptualization.

3.3 The notion of an ontology base

This section introduces the notion of ontology base. An ontology base [M99a] is a knowledge representation model for capturing domain axiomatizations. This notion is used as a core component in the DOGMA project.

Basically, an ontology base consists of a set of lexons. A lexon is a binary relationship between context-specific linguistic terms, or in other words, a lexical rendering of a binary conceptual relation.

3.3.1 Definition (Lexon)

A lexon is described in [M99a][M99b] as a tuple of the form:

$$< \gamma : T_1, r, r', T_2 >$$

Where:

γ is a context identifier.

T_1 and T_2 are linguistic terms from a language L.

r and r' are lexicalizations of the pair roles of a binary conceptual relationship R; the role r' is the inverse of the role r. One can verbalize a lexon as (T_1 r T_2), and (T_2 r' T_1). For example, the pair roles of a *subsumption* relationship could be: "Is a type of" and "Has type"; the pair roles of a *parthood* relationship could be: "is a part of" and "has part", and so forth.

The following is a set of lexons, as a simple example of an ontology base:

<Commerce: Person, Issues, Issued by, Order>
<Commerce: Order, Settled Via, Settles, Payment Method>
<Commerce: Money Order, Is a type of, Has type, Payment Method>
<Commerce: Check, Is a type of, Has type, Payment Method>
<Commerce: Payment Card, Is a type of, Has type, Payment Method>
<Commerce: Credit Card, Is a type of, Has type, Payment Card>
<Commerce: Credit Card, Has, Is of, Expiration Date>

-D

3.3.2 Definition (Concept)

A term T within a context γ is assumed [M99a] to refer to *a concept* C:

$$(\gamma, T) \rightarrow C$$

Notice, for example, that within the context 'Commerce', the linguistic term 'Order' refers to "A commercial document used to request someone to supply something in return for payment". It may refer to other concepts within other contexts, e.g. within the context 'Military', the term 'Order' refers to "A command given by a superior that must be obeyed"[43]. Further detail about the notion of context will be discussed in the next section.

As we have discussed earlier, a concept is a set of rules in our mind about a certain thing in reality. The notion of *intended meaning* (or word meaning/sense) can be used alternatively with the notion of concept to denote something. The set of all possible instances (i.e. in all possible stats of affairs) that comply with these rules are called *intended models*.

3.3.3 Definition (Role)

A role within a context is not intended to refer to a concept; thus, $(\gamma, r) \rightarrow C$ is improper. In other words, our notion of role does not refer to a "stand alone" unary (or binary) concept. Rather, roles only lexicalize the participation of a "unary concept" in an n-ary conceptual relationship. As the notion of a lexon is a lexical rendering of a binary conceptual relationship, we formalize a lexon as two context-specific terms playing mutual roles, that both refers to a *binary concept (typically called binary conceptual relation)*:

$$< (\gamma, T, r), (\gamma, T, r) > \rightarrow C^2$$

The notation of a context-specific term playing a role (γ, T, r) is called *concept-role.*

[43] These two definitions of the term "Order" are taken from WordNet, (May 2004) http://www.cogsci.princeton.edu/cgi-bin/webwn.

For practical purposes, we shall not require for both roles to be explicitly lexicalized within a lexon. We assume that at least one role is to be lexicalized, such as <Bibliography, Book, is-a, Written Material>.

An ontology base is intended to capture binary relationships. This does not deny the existence of ternary (or more) relationships. We believe that relationships in practice are mainly binary. Moreover, binary relations are easier for ontology builders to model, extract, or reason with.

3.3.4 Definition (Mapping lexons into first order logic)

Each lexon $<\gamma{:}\mathrm{T}_1,r,r',\mathrm{T}_2>$ in the ontology base is mapped into three statements in first order logic, as the following[44]:

$$\forall x\, T_1(x) \rightarrow (\forall y\, r(x,y) \rightarrow T_2(y))$$

$$\forall y\, T_2(y) \rightarrow (\forall x\, r'(y,x) \rightarrow T_1(x))$$

$$\forall x.y\ r(x,y) \leftrightarrow r'(y,x)$$

For example, the mapping of the lexon <Commerce: Person, Issues, IssuedBy, Order> into first order logic can be done as follows:

$$\forall x\ Person(x) \rightarrow (\forall y\ Issues(x,y) \rightarrow Order(y))$$

$$\forall y\ Order(y) \rightarrow (\forall x\ IssuedBy(y,x) \rightarrow Person(x))$$

$$\forall x.y\ Issues(x,y) \leftrightarrow IssuedBy(y,x)$$

Notice that Context is not part of our formal mapping of lexons. As we shall discuss in the next section, a context is an *informal* notion used to bound the interpretation of a linguistic term into a concept. Linguistic terms, e.g. 'Person', 'Order', etc. can be seen as unambiguous terms (i.e. concepts) within the lexon formal mapping. A lexon (or it formal mapping) is assumed to be true (i.e. axiom) within its context, see section

[44] This mapping was achieved over the course of a fruitful discussion with Stijn Heymans.

3.3.5. In section 3.3.7 we shall discuss how to introduce further formal axiomatizations at the ontology base level, for targeting systematic ontological quality.

Finally, our formal lexon mapping assumes unique role names. Each role label (or InvRole) should be unique within the formal mapping of lexons. As this is might not be the case in practice, one can provide an "internal" naming convention, for example, by renaming 'Issues' as 'Issues_Order' and 'IssuedBy' as 'IssuedBy_Person'.

At this point, we have established how that lexons are the basic building blocks of an ontology base and that they are the basic domain axioms. *The principal role of an ontology base is to be a shared vocabulary space for application axiomatizations.* As sharing lexons means sharing the same concepts and their intended models, semantic interoperability between classes of autonomous applications can be achieved, basically, by sharing a certain set of lexons[45].

3.3.5 The notion of context

The notion of context has been, and still is, the subject of occasionally intense study, notably in the field of Artificial Intelligence. It has received different interpretations. Commonly, the notion of context has been realized as a set of formal axioms (i.e. a theory) about concepts. It has been used among other things: to localize or encode a particular party's view of a domain, cf. C-OWL [BHGSS03]; as a background, microtheory, or higher-order theory for the interpretation of certain states of affairs [M93][S00][MVBCFGG04][SGP98][GG0]; and to facilitate the translation of facts from one context to another, as in KIF [PFP+92].

In our approach, we shall use the notion of context to play a "*scoping" role* at the ontology base level. We say a term *within* a context refers to a

[45] As we shall show in section 3.4, a class of interoperating applications may need to agree on and share some rules that constrain the use of a concept, i.e. share the same legal models.

concept, or in other words, *that context is an abstract identifier that refers to implicit (or maybe tacit*[46]*) assumptions, in which the interpretation of a term is bounded to a concept.*

Notice that a context in our approach *is not explicit* formal knowledge. In practice, we define context by referring to a source (e.g. a set of documents, laws and regulations, informal description of "best practice", etc.), which, by *human understanding*, is assumed to "contain" those assumptions. Lexons are assumed (by human understanding) to be "true within their context's source". Hence, a lexon is seen as a domain *axiom*.

In section 6.2.1, we suggest some "best practices" for defining a context. In section 7.2.1, we present an example of a context definition in a real-life case study. The lessons we learnt and our experience with defining contexts are also reported in this section.

Before proceeding to discuss further formal axiomatizations at the ontology base level, we introduce the notion of *gloss* as part of the ontology base model.

3.3.6 The notion of Gloss[47]

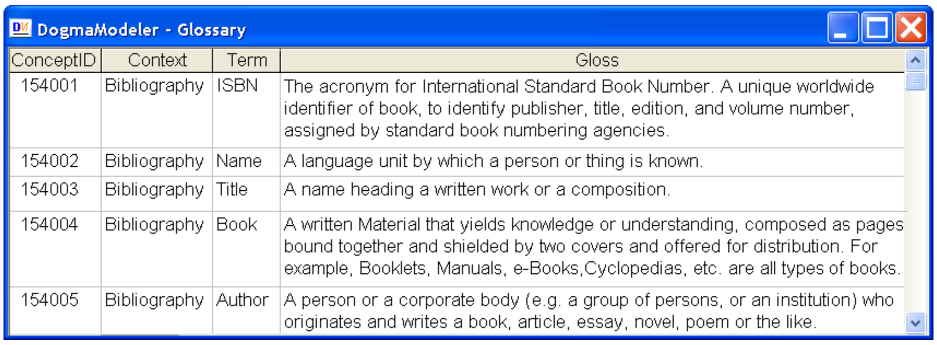
Within an ontology base, each combination of a Context and a Term is given a unique number, called a *ConceptID*. Thus, one can alternatively use ConceptID or (Context, Term) to uniquely refer to a concept[48] within an ontology base.

[46] The difference between implicit and tacit assumptions, is that the implicit assumptions can, in principle, be articulated but still they have not, while tacit assumptions are the knowledge that cannot be articulated. it consists partially of technical skills -the kind of informal, hard-to-pin-down skills captured in terms like "know-how", and "we know more than we can tell or put in words". However, even though tacit assumptions cannot be articulated, they can be transferred through other means over than verbal or formal descriptions [Inn+03] [N94].

[47] Later this section was revised and extended (See [J06]).

[48] For some approaches, e.g. [KTT03], the lexicalization of concepts is not necessary - concepts can be represented and referenced only by ConceptIDs. In our approach however, this is not allowed. Each concept must be *lexicalized* by a linguistic term.

Each concept should be described by a *gloss*. A gloss is an auxiliary *informal account* for the commonsense perception of humans of the intended meaning of a linguistic term. See fig. 3.5.


DogmaModeler - Glossary

ConceptID	Context	Term	Gloss
154001	Bibliography	ISBN	The acronym for International Standard Book Number. A unique worldwide identifier of book, to identify publisher, title, edition, and volume number, assigned by standard book numbering agencies.
154002	Bibliography	Name	A language unit by which a person or thing is known.
154003	Bibliography	Title	A name heading a written work or a composition.
154004	Bibliography	Book	A written Material that yields knowledge or understanding, composed as pages bound together and shielded by two covers and offered for distribution. For example, Booklets, Manuals, e-Books,Cyclopedias, etc. are all types of books.
154005	Bibliography	Author	A person or a corporate body (e.g. a group of persons, or an institution) who originates and writes a book, article, essay, novel, poem or the like.

Fig. 3.5. A list of concepts described by glosses.

Notice that the information provided in a gloss can be translated, in principles, into formal logical statements. However, both are seen and used in complement rather than as alternatives.

The purpose of a gloss *is not* to provide or catalogue general information and comments about a concept, as conventional dictionaries and encyclopedias do [MBFGM90]. A gloss, for formal ontology engineering purposes, is supposed to render factual knowledge that is critical to understanding a concept, but that is unreasonable or very difficult to formalize and/or articulate explicitly.

The following are some guidelines to consider when deciding what should and should not be provided in a gloss.

1. It should start with the *principal/super type* of the concept being defined. For example, "Search engine: A computer program that …", "Invoice: A business document that…", "University: An institution of …".

2. It should be written in the form of propositions, offering the reader *inferential knowledge* that helps him to construct the image of the

-D

concept. For example, instead of defining 'Search engine' as "*A computer program for searching the internet", it can be defined as, "One of the most useful aspects of the World Wide Web. Some of the major ones are Google, Galaxy….*". One can also say "*A computer program that enables users to search and retrieve documents or data from a database or from a computer network…*".

3. More importantly, it should focus on distinguishing characteristics and intrinsic properties that differentiate the concept from other concepts. For example, compare the following two glosses of a 'Laptop computer': (1) "*A computer that is designed to do pretty much anything a desktop computer can do. It runs for a short time (usually two to five hours) on batteries*"; and (2) "*A portable computer small enough to use in your lap…*". Notice that according to the first gloss, a 'server computer' running on batteries can be seen as a laptop computer; also, a 'Portable computer' that is not running on batteries is not a 'Laptop computer'.

4. The use of supportive examples is strongly encouraged: (1) to clarify true cases that are commonly known to be false, or false cases that are known to be true; and (2) to strengthen and illustrate distinguishing characteristics (by using examples and counter-examples). The examples can be types and/or instances of the concept being defined. For example: "Legal Person: *An entity with legal recognition in accordance with law. It has the legal capacity to represent its own interests in its own name, before a court of law, to obtain rights or obligations for itself, to impose binding obligations, or to grant privileges to others, for example as a plaintiff or as a defendant. A legal person exists wherever the law recognizes, as a matter of policy, the personality of any entity, regardless of whether it is naturally considered to be a person.*

Recognized associations, relief agencies, committees and companies are examples of legal persons".

5. It should be consistent with the lexons and formal definitions.
6. It should be sufficient, clear, and easy to understand[49].

Glosses play a significant role during the ontology development, deployment, and evolution phases. As we discussed in section 2.3, ontologies are being developed, reviewed, used, and maintained by many different people over different times and locations. Indeed, glosses are easier to understand and agree on than formal definitions, especially for non-intellectual domain experts. Glosses are a useful mechanism for understanding concepts *individually* without needing to browse and reason on the position of concepts within an axiomatized theory. Further, compared with formal definitions, glosses help to build a "deeper" intuition about concepts, by denoting to implicit or tacit assumptions.

Hence, we fulfill the R3 requirement: critical assumptions that make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology, even if informally, to facilitate both users' and developers' commonsense perception of the subject matter.

3.3.7 Further formal axiomatizations (Incorporating upper level ontologies)

In order to achieve *a systematic ontological quality and precision*[50] on the specification of the intended meanings of linguistic terms, these specifications might need to receive more formal restrictions, than just mapping lexons into logical statements.

[49] There is more to say on how to define a gloss; we limited ourselves in this thesis to present the most relevant issues.
[50] The notion of "ontological precision" is defined by Aldo Gangemi in [G04] as "*the ability to catch all and only the intended meaning*".

For example, without introducing further formal restrictions to the following lexons:

```
<Bibliography: Man, Is-a, Person>
<Bibliography: Author, Is-a, Person>
<Bibliography: Mustafa, Is-a, Person>
```

The ontological difference (or the misuse of 'is-a') cannot be systematically detected[51].

In this section, we discuss how a formal axiomatic system can be introduced into an ontology base.

As we have chosen to represent formal domain axiomatization in a data model (i.e. ontology base), arbitrary and expressive formal definitions are restricted (see our discussion on this issue in section 3.2.3). Therefore, we extend the ontology base model to incorporate primitives of upper level ontologies. Our incorporation of upper level ontologies in this thesis is fairly simplistic; deep philosophical argumentations that are necessary for such incorporation are presented schematically for the sake of simplicity. It is important to note that the upper ontologies are still very much works in progress. We have chosen to incorporate the topic in this thesis for the sake of contextual completeness as we believe that it complements the general idea of our approach.

Upper level ontologies are formal axiomatic systems that describe the most general categories of reality. Such ontologies are not only application and task independent but also domain (and possibly language) independent axiomatizations [DHHS01] [G98b].

Based on the literature of upper level ontologies as found for example in [DHHS01] [G98b] [MBGGO03], *we introduce,* in our approach, *the*

[51] By assuming that the 'is-a' refers to a subsumption relationship (i.e. Sub-Type of), only the first lexon is correct. The 'is-a' in the second lexon should interpreted as "is role of", because 'Author' is a role of 'Person' and not a type of a 'Person'; and obviously, the last lexon refers to 'is instance of'. See [GW02] for more details on this issue.

notion of upper-form. Each term within a context should have an upper-form, likewise, each lexon should have an upper-form.

Term upper-forms

Term upper-forms are superior types of concepts, such as substantial, feature, abstract, region, event, process, type, role, particular, etc. The notation of term upper-form is:

$$\gamma(T) : < UpperFormName >$$

For example, Bibliography(Person):Substantial, Bibliography(Author):Substantial, Bibliography(First-Name):Property, etc.

A term can have several upper-forms; the notation: $\gamma(T) : \{UpperForm\}$. For example, Bibliography(Person):{Substantial, Type}, Bibliography(Author):{Substantial, Role}, Bibliography(Mustafa):{Substantial, Instance}, etc.

Lexon upper-forms

Lexon upper-forms are relationship kinds, also called "basic primitive relations" [MBGGO03], such as parthood, dependence, property-of, attribution, subsumption, etc. Such relationship kinds are carefully and formally axiomatized in upper level ontologies, and they are general enough to be applied in multiple domains. Our notation of a lexon upper-form is:

$$< \gamma : \mathrm{T}_1, r, r', \mathrm{T}_2 > : < UpperFormName >$$

For example, the lexon "<Bibliography: Book, Is-a, HasType, Written Material>: Subsumption" is a subsumption relationship where the concept 'Book' formally subsumes the concept 'Written Material'. The lexon "<Bibliography: Book, Has-Part, Is-Part-Of, Chapter>: Parthood" is a parthood relationship, where an instance of the concept 'chapter' is a part of an instance of the concept 'Book'. The lexon "<Bibliography: Author, Has, Is-Of, Name>: Property" is a property-of relationship, where the concept 'Name' is a property of the concept 'Author', and so forth.

The idea of introducing upper-forms is to bring on or *induce* the formal axiomatization of such relation kinds, as defined in upper level ontologies, into lexons. In other words, upper-forms are used as theoretical tools to incorporate *formal account* into lexons. For example, the formal account of the lexon "<Bibliography, Mustafa, instance-of, Author>: Instantiation" is induced by the formal axiomatization of the instantiation relationship as found [GGMO01], see fig 3.6.

$$I(x,y) \rightarrow \neg I(y,x) \qquad (asymmetry)$$
$$(I(x,y) \wedge I(x,z)) \rightarrow (\neg I(y,z) \wedge \neg I(z,y) \qquad \text{(antitransitivity)}$$
$$Particular(x) =_{def} \neg \exists y(I(y,x))$$
$$\text{Universal}(x) =_{\text{def}} \neg Particular(x)$$

Fig. 3.6. A formal axiomatization of the instantiation relationship, as found in [GGMO01].

The formal account of the lexon "<Bibliography: Book, Has-Part, Is-Part-Of, Chapter>: Parthood" is induced by the formal axiomatization of the parthood relationship as found in [GGMO01], see fig 3.7.

$$P(x,x)$$
$$(P(x,y) \wedge P(y,x)) \rightarrow x = y$$
$$(P(x,y) \wedge P(y,z)) \rightarrow P(x,z)$$

Fig. 3.7. A formal axiomatization of the Parthood relationship as found in [GGMO01].

By inducing the formal axiomatization of the 'Subsumption' relationship, as found in [GGMO01], the following lexon is incorrect because a 'Role' cannot subsume a 'Type'.

Bibliography(Person):{Substantial, Type}
Bibliography(Author):{Substantial, Role}
<Bibliography: Author, Is-a, Person>: Subsumption

-D

Notice that formal axiomatizations of such upper forms are not necessary to be used at runtime by applications that use or share lexons. The main goal is to use these axiomatizations as theoretical tools to achieve a systematic quality at the development and maintenance time of an ontology.

Our methodological principles and their implementation prototypes are independent of a particular upper level ontology. The choice of which upper level ontology to use is left to ontology builders. In an upcoming effort, we plan to develop a library of upper-ontology components, so that ontology builders will be able to plug-in and automatically reason about the quality of their lexons.

3.4 Application axiomatization

In the previous sections, we have presented and discussed the first part of the ontology double articulation principle. We have introduced the notion of an ontology base for capturing domain axiomatizations independently of usability perspectives. In this section, we introduce the second part of the ontology double articulation principle: application axiomatizations. First, we discuss the general properties of these axiomatizations; then, we introduce the notion of application ontological commitments.

While the axiomatization of domain knowledge is mainly concerned with the characterization of the "intended models" of concepts, the axiomatization of application knowledge is mainly concerned with the characterization of the "legal models" of these concepts (see fig. 3.4). Typically, as domain axiomatizations are intended to be shared, public, and highly reusable at the domain level, application axiomatizations are intended to be local and highly usable at the task/application-kind level.

As we have discussed earlier, applications that are interested only on a subset of the intended models of a concept (according to their usability perspective) are supposed to provide some rules to specialize these intended models. Such a specialization is called an *application axiomatization*. Notice that this specialization is not seen as two different concepts subsuming one another through a “subsumption relationship”. Rather, the vocabulary -of unary and binary concepts- used in application axiomatization is restricted to the vocabulary defined in its domain axiomatization. As shall be cleared later in this section, an application axiomatization becomes a set of rules to constrain a certain use of domain vocabulary. Formally speaking, these rules declare what should necessarily hold in any possible world for a class of applications.

A particular application commits to the intended meaning of a domain vocabulary (i.e. in an ontology base) through its application axiomatization. This commitment is called application’s *ontological commitment*. An application axiomatization typically consists of: (1) an *ontological view* that specifies which domain concepts in an ontology base are relevant to include and represent in this axiomatization. These concepts can be explicit lexons or derived from lexons, (2) a set of rules to characterize the legal models of the ontological view, i.e. to formally declare what should necessarily hold in any possible world for the applications sharing this axiomatization.

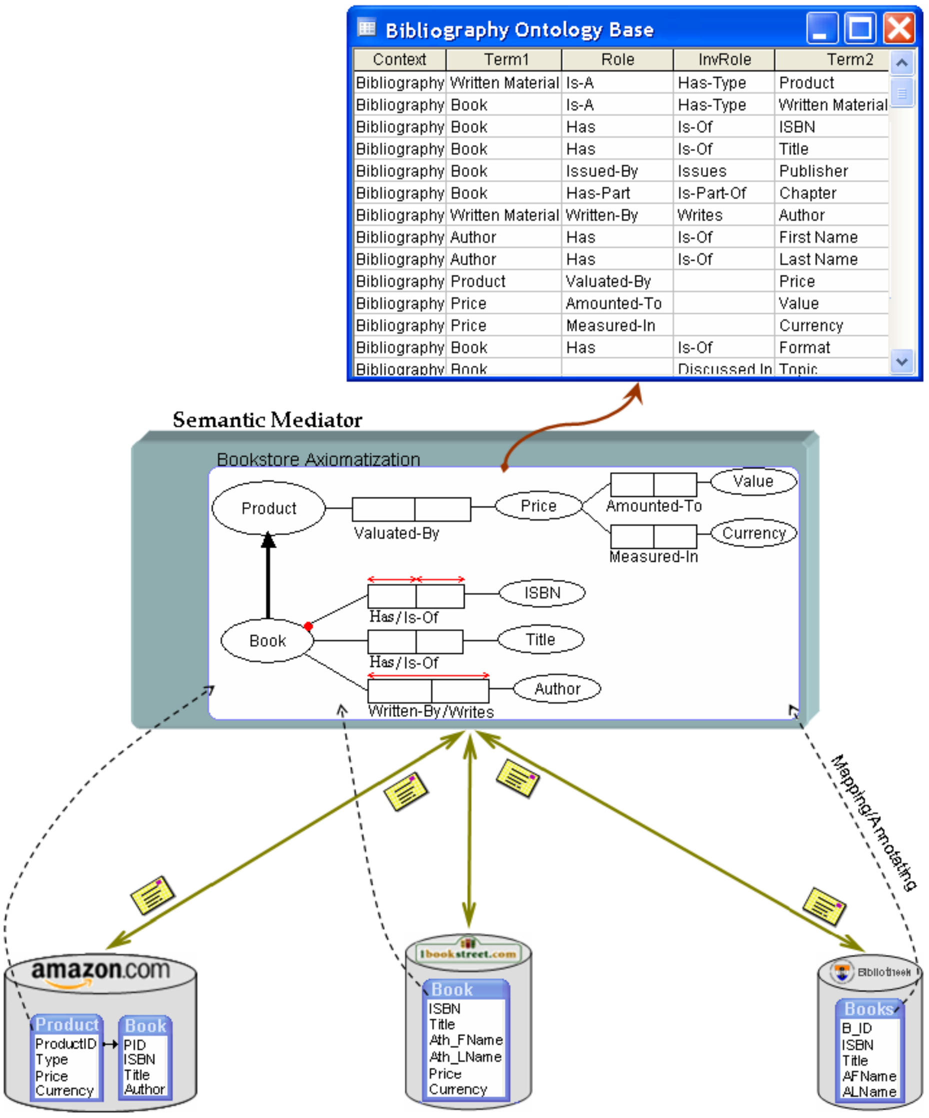
We say that a particular extension of an application (i.e. a set of instances) commits to an ontology base through an application axiomatization if it conforms to or is consistent with the ontological view and the rules declared in this axiomatization (cf. model-theoretic semantics). We shall came back to this issue in section 4.4.2.

3.4.1 Example

This example is based on that presented in section 3.1.2. We show an application scenario of software agents interoperating through a semantic mediator to exchange data messages and business transactions. The interoperation is enabled by the sharing of the same Bookstore axiomatization, i.e. as a global and legal data model[52]. The data source (or its "export schema" [ZD04]) of each agent is mapped into the shared axiomatization. All exchanged data messages (e.g. those formed in XML, RDF, etc.) can be validated according to whether they conform to the rules and the ontological view declared in the Bookstore axiomatization by using for example model-theoretic semantics [R88].

[52] This way of sharing and using axiomatizations (as global schema) seems more applicable to data integration and mediation systems [BB04][ZD04][CBB+04]. They can also be used to describe web services [NM02]. For example, an axiomatization could be specified for each web service (to describe the "static" information provided to/by a web service), so that all agents accessing a web service share the same axiomatization.

-D



Bibliography Ontology Base

Context	Term1	Role	InvRole	Term2
Bibliography	Written Material	Is-A	Has-Type	Product
Bibliography	Book	Is-A	Has-Type	Written Material
Bibliography	Book	Has	Is-Of	ISBN
Bibliography	Book	Has	Is-Of	Title
Bibliography	Book	Issued-By	Issues	Publisher
Bibliography	Book	Has-Part	Is-Part-Of	Chapter
Bibliography	Written Material	Written-By	Writes	Author
Bibliography	Author	Has	Is-Of	First Name
Bibliography	Author	Has	Is-Of	Last Name
Bibliography	Product	Valuated-By		Price
Bibliography	Price	Amounted-To		Value
Bibliography	Price	Measured-In		Currency
Bibliography	Book	Has	Is-Of	Format
Bibliography	Book		Discussed In	Topic

Fig. 3.8. Meaningful semantic interoperation between Bookstore applications.

The ontological view of the above bookstore axiomatization specifies which concepts are relevant for the task(s) of this application scenario. These concepts correspond to explicit lexons in the ontology base, or they might be derived from these lexons. One can see in the ontology base that a 'Book' is not explicitly a 'subtype of' a 'Product' as specified in the

-D

Bookstore axiomatization. This subsumption is derived from these lexons: {<Bibliography: Book, Is-A, Written Material>, <Bibliography: Written Material, Is-A, Product>}. Based on these subsumptions, some inheritance also might be drawn; For example, 'Book' inherits the relationship <Bibliography: Book, Written-By, Author> from its 'Written Material' supertype. The choice of which concepts and relations should be included in an axiomatization is an application-dependent issue or subject to a usability perspective. See our discussion on this issue in section 2.2.

In this bookstore axiomatization, four rules are declared and can be *verbalized* as: 1) each Book must Has at least one ISBN; 2) each Book Has at most one ISBN; 3) each ISBN Is-Of at most one Book; 4) it is possible for a Book to be Written-by several Authors, and it is possible for an Author to write several Books.

Notice that the double articulation principle enables usability perspectives to be encountered and encoded outside domain axiomatization. In turn, this indeed increases the usability of application axiomatizations as well as increases the reusability of domain axiomatization.

Depending on the application scenario, application axiomatizations may be used in different ways. For example, in the Semantic Web and information search/retrieval scenarios, declaring rules might be not important because the main idea of these scenarios is to expand (rather than to constrain) queries. Filtering the unwanted results (i.e. illegal models) is the responsibility of the people who usually are involved in such application scenarios[53]. In chapter 7, we show the application scenario of an ontology-based user interface, where application axiomatizations are used as shared data models of complaint web forms.

To increase usability of application axiomatizations, they might be specified in multiple specification languages, such as DAML+OIL, OWL,

[53] For example, as Google users filter out the unwanted web-pages that appear as a result of their search.

RuleML, EER, UML, etc. Figure 3.9 shows the above Bookstore axiomatization expressed in OWL.

```
  .
<owl:Class rdf:ID="Product" />
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf rdf:resource="#Product" />
</owl:Class>
<owl:Class rdf:ID="Price" />
<owl:Class rdf:ID="Value" />
<owl:Class rdf:ID="Currency" />
<owl:Class rdf:ID="Title" />
<owl:Class rdf:ID="ISBN" />
<owl:Class rdf:ID="Author" />
<owl:ObjectProperty rdf:ID="Valuated-By">
<rdfs:domain rdf:resource="#Product" />
<rdfs:range  rdf:resource="#Price" />
</owl:ObjectProperty>
<owl:DataProperty rdf:ID=" Amounted-To .Value">
  <rdfs:domain rdf:resource="#Price" />
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:ObjectProperty>
<owl:DataProperty rdf:ID="Measured-In.Currency">
  <rdfs:domain rdf:resource="#Price" />
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:ObjectProperty>
<owl:DataProperty rdf:ID="Has.ISBN">
  <rdfs:domain rdf:resource="#Book" />
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer "/>
</owl:ObjectProperty>
<owl:DataProperty rdf:ID="Has.Title">
  <rdfs:domain rdf:resource="#Title" />
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Written-By">
  <owl:inverseOf rdf:resource="#Writes "/>
  <rdfs:domain rdf:resource="#Book" />
  <rdfs:range  rdf:resource="#Author" />
</owl:ObjectProperty>
<owl:Restriction>
  <owl:onProperty rdf:resource="# Has.ISBN " />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
  .
```

-D

Fig. 3.9. An OWL representation of the Bookstore ontological commitment.

Although both representations share the same intended meaning of concepts at the domain (/ontology base) level, notice the disparities between ORM and OWL in representing the Bookstore axiomatization. For example, ORM does not distinguish between DataProperties and ObjectProperties as does OWL. This is an example of an epistemological difference[54]. The ORM uniqueness constraint that spans over "Written-By/Writes" cannot (or should not) be expressed in OWL, as it is implied by definition[55]. The other uniqueness and mandatory constraints are expressed as a one cardinality restriction in OWL.

Such logical and epistemological disparities (which are induced by the difference between the formalizations and the constructs of both languages) illustrate different ways of characterizing the legal models of application axiomatizations. The choice of which language is more suitable for specifying application axiomatizations depends on the application scenario and perspectives. For example, ORM and EER are mainly suitable for database and XML (-based) application scenarios since they are comprehensive in their treatments of the integrity of data sets. For inference and reasoning application scenarios, description logic based languages (such as OWL, DAML, etc.) seem to be more applicable than other languages, as they focus on the expressiveness and the decidability of axioms.

Allowing different languages, optimized techniques, or methodologies to be deployed at the application axiomatization level will indeed increase the usability of these axiomatizations. A recent application axiomatization language called Ω-RIDL [VDM04] has been developed within the

[54] See the definition of "epistemological" in appendix D.
[55] The formalization of ObjectProperties in OWL does not allow the same tuple to appear twice in the same set, such as Written-By = {<author1, book1>, < author1, book1>,…}.

DOGMA framework. Its creators claim it is better suited to the database applications' commitment to an ontology base.

3.5 Discussion

In this chapter, we have presented the double articulation principle. We have shown how application verses domain axiomatizations can be well articulated. We have introduced the notion of an ontology base for capturing domain axiomatizations, and the notion of application axiomatizations by which particular applications commit to the intended meaning of domain vocabulary.

In the following paragraphs, we summarize the main advantages of the double articulation principle:

- *Increase reusability of domain axiomatization, as well as usability of application axiomatizations*. As we have shown in this chapter, the application-independence of an ontology is increased by separating domain and application axiomatizations. Usability perspectives have a neglectable influence on the independence of a domain axiomatization, because ontology builders are prevented from encoding their application-specific axioms. In other words, domain axiomatizations are mainly concerned with the characterization of the "intended models" of concepts, while application axiomatizations are mainly concerned with the characterization of the "legal models" of these concepts. **Hence, we fulfill the R2 engineering requirement**: The influence of usability perspectives on ontology axioms should be well articulated, in pursuit of both usability and reusability.
- *Allows different communities to create and maintain domain axiomatization (typically public) and application axiomatizations (typically local).* Indeed, domain experts, lexicographers, knowledge engineers, and even philosophers, may contribute to

the development, maintenance, and review phases of domain axiomatizations. It is needless for them to know why and how these axiomatizations will be used. Application-oriented experts can also contribute to and focus on the development phases of application axiomatizations, without needing to know about the correctness of domain axioms. **Hence, we fulfill the R4 engineering requirement**: the ontology representation model should be capable of distributed and collaborative development.

- *Allows the deployment of differently optimized technologies and methodologies to each articulation*. For example, relational database management systems can be used (with high scalability and performance) to store and retrieve large-scale ontology bases. Natural language parsing and understanding techniques can be employed for extracting lexons from texts (see [PSDM03] for an example of preliminary results on this issue). Different specification languages can be used to specify application axiomatizations and these increase the usability of these axiomatizations.

Furthermore, *the importance of linguistic terms in ontology engineering is observed and incorporated* in our approach. Not coincidentally, our approach allows for the adoption and reuse of many available lexical resources to support (or to serve as) domain axiomatizations. Lexical recourses (such as lexicons, glossaries, thesauruses and dictionaries) are indeed important recourses of domain concepts. Some resources focus mainly on the morphological issues of terms, rather than categorizing and clearly describing their intended meanings. Depending on its description of term meaning(s), its accuracy, and maybe its formality[56], a lexical resource can play an important role in ontology engineering.

[56] i.e., the discrimination of term meanings in a machine-referable manner.

An important lexical resource that is organized by word meanings (i.e. concepts, or called synsets) is WordNet [MBFGM90]. WordNet offers a machine-readable and comprehensive conceptual system for English words. Currently, a number of initiatives and efforts in the lexical semantic community have been started to extend WordNet to cover multiple languages. As we have discussed in section 3.2.2, the consensus about domain concepts can be gained and realized by investigating these concepts at the level of a human language conceptualization. This can be practically accomplished e.g. by adopting the informal description of term meanings that can be found in lexical resources such as WordNet, *as glosses*. We shall illustrate this issue in our implementation prototype in chapter 6.

The notion of gloss as an auxiliary informal account of the intended meaning of a linguistic term **fulfills the R3 engineering requirement**: critical assumptions that make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology, even if informally, to facilitate both users' and developers' commonsense perception of the subject matter.

In the next chapter, we proceed to present the second methodological principle for ontology engineering: the ontology modularization principle.

-D

Chapter 4

Ontology Modularization

"Modularity is a key requirement for large ontologies in order to achieve re-use, maintainability, and evolution."

- (Alan Rector, [R03])

This chapter presents the second engineering principle of our approach: Ontology Modularization. In section 4.1, we introduce and illustrate the general idea of the ontology modularization principle. Section 4.2 overviews other approaches to ontology modularization. We describe our approach to modularity and composition and present the formal details in sections 4.3 and 4.4 respectively. As an illustration of our approach, in section 4.5 we present an algorithm for the automatic composition of modules specified in ORM. Section 4.6 summarizes the main advantages gained and the engineering requirements fulfilled by the modularization principle.[57]

[57] Later, this section was revised and extended, see [J05a].

4.1 Introduction

In this section, we introduce and illustrate the general idea of the ontology modularization principle. Further details follow in the next sections.

The modularization principle aims to fulfill the following ontology engineering requirements:

- **R1.** Ontologies should be engineered in a way that allows for the isolation and identification of the reusable parts of the ontology.
- **R4.** The ontology representation model should be capable of distributed and collaborative development.
- **R5.** Ontologies should be engineered in a way that enables smooth and efficient evolution.
- **R6.** Ontologies should be engineered in a way that allows easy replacement of the axiomatization of ontology parts.

The main idea of the modularization principle is to decompose an application axiomatization into a set of smaller related modules, which: 1) are easier to reuse in other kinds of applications; 2) are easier to build, maintain, and replace; 3) enable distributed development of modules over different locations and expertise; 4) enable the effective management and browsing of modules, e.g. enabling the construction of libraries of application-kind axiomatizations.

To compose modules, we propose a composition operator: all atomic concepts and their relationships (i.e. lexons) and all constraints, across the composed modules, are combined together to form one axiomatization (called modular axiomatization).

4.1.1 A simple example

In what follows, we give an example to illustrate the (de)composition of application axiomatizations. Fig. 4.1 shows two axiomatizations of Book-

-D

Shopping and Car-Rental applications, defined on an e-commerce ontology base[58]. Notice that both axiomatizations share the same axioms about the "payment" conceptualization.

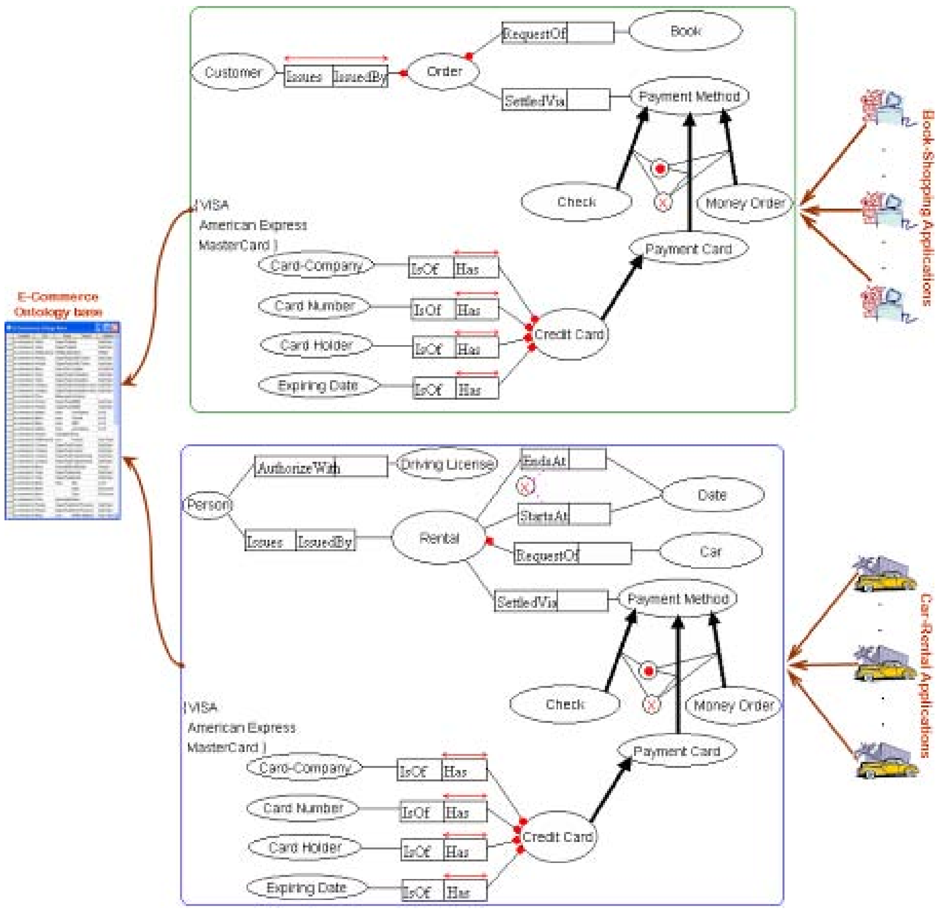


Fig. 4.1. Book-shopping and Car-Rental axiomatizations.

Instead of repeating the same effort to construct the axiomatization of the "payment" part, the modularization principle suggest that we decompose these axiomatizations into three modules, which can be shared and reused among other axiomatizations (see fig. 4.2). Each application-type (*viz.* Book-Shopping and Car-Rental) selects appropriate modules (from a

[58] The e-commerce ontology base is not illustrated here for the sake of brevity.

library of application axiomatizations) and composes them through a composition operator. The result of the composition is seen as one axiomatization[59].

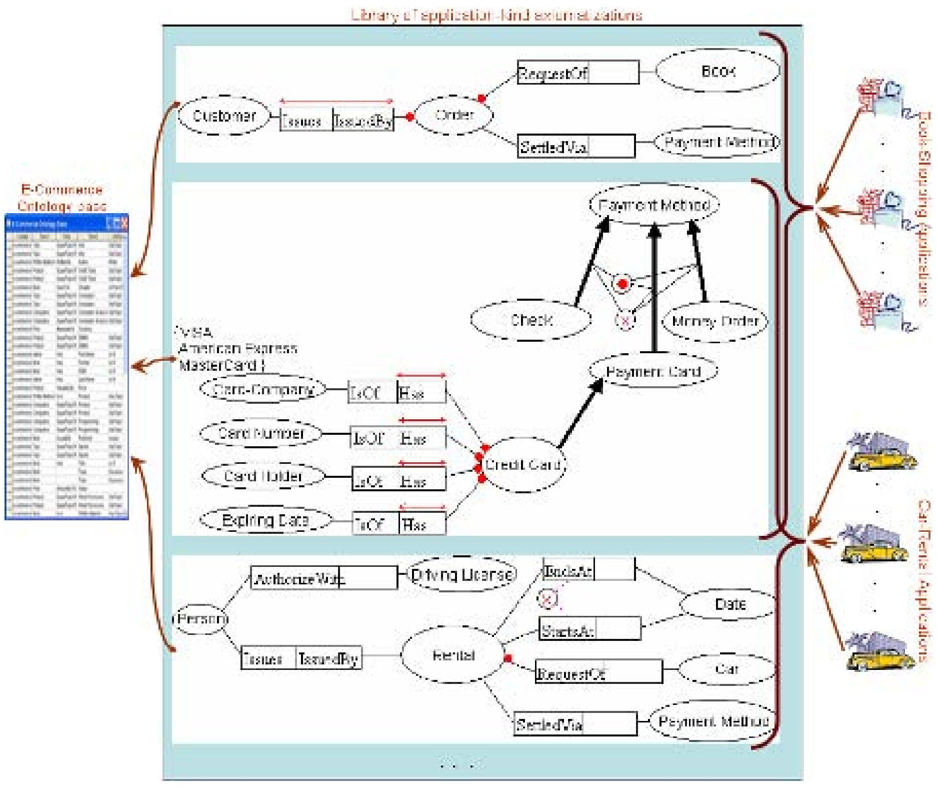


Fig. 4.2. Modularized axiomatizations.

Engineering application axiomatizations in this way will not only increase their reusability, but also the maintainability of these axiomatizations. As the software engineering literature teaches us, small modules are easier to understand, change, and replace [P72] [SWCH01]. An experiment by [BBDD97] proves that the modularity of object-oriented design indeed enables better maintainability and extensibility than structured design.

[59] The illustrated composition in this example is very simplistic, as each pair of modules overlap only in one concept, i.e. the "Payment Method". In farther sections, we discuss more complicated compositions, in which rules in different modules may contradict or imply each other.

-D

Decomposing axiomatizations into modules also enables the distributed development of these modules over different location, expertise, and/or stakeholders. As an analogy, compare the capability of distributing the development of a program built in Pascal with a program built in JAVA, i.e. structured verses modular distributed software development. In chapter 7, we report our practical experience and the maintainability in the distributed development of a Customer Complaint Ontology (CContology).

4.2 Related work

The importance of modularity has received limited attention from within the knowledge representation community [SK03]. Modularity has been adopted by some researchers to achieve more scalability for reasoning and inference services. A knowledge base is seen as a set of distributed knowledge bases, with each base referred to as a module. In this way reasoning is performed locally in each module, and the results are propagated toward a global solution. Global soundness and completeness (i.e. consistency) follows from the soundness and completeness of each local reasoner [WSG+04]. The performance of such reasoning is claimed to be linear in the tree structure in most cases [AM04].

Borgida and Serafini have proposed in [BS03] an extension to description logics to enable more sophisticated distributed reasoning. Objects in distributed and autonomous data sources are connected through complex mappings. The authors claim that these mappings form a "global view" of the connected data sources.

In [SK03] [SH05], Stuckenschmidt and Klein have proposed an approach to ontology modularization similar to view-based data integration. A data source (i.e. a schema and it instances) is seen as a module. All modules, as such, are connected by conjunctive queries. The result of each mapping query is computed and added as an axiom to the module using the result.

-D

Reasoning in a module depends on the answer sets of the queries used to connect it to other modules. A modular ontology in this approach is defined as a set of modules that are connected by external concepts and relation definitions.

A quite similar approach to the previous one is proposed by Oberle and colleagues [VOS03] who defined a view language for connecting RDF resources to each other.

A recent survey on distributed and modular knowledge representation (towards scalable reasoning) can be found in [WSG+04].

While the approaches described above are concerned with the modularity at the *deployment* phase of ontologies (i.e. distributed reasoning), Rector [R03] has proposed another approach to modularity that is mainly concerned with the distributed *development* of the TBox of an ontology. Rector's proposal is to decompose an ontology into a set of independent disjoint skeleton taxonomies restricted to simple trees. Disjoint taxonomies (i.e. modules) can then be composed using definitions and relationships between concepts in the different modules. In contrast to other approaches, the result of such a composition can be seen as one local TBox. This approach is motivated by Guarino's analyses of types [G98b]. Assuming that each type has a distinct set of identity criterion, when a type specializes another type, it adds further identity criterion to those carried by the subsuming type. The taxonomy of such types is always a tree.

4.3 Our approach

In this section we introduce our approach to ontology modularization and composition on an abstract level. The formal and technical details will be provided in the following sections.

In our approach, we are mainly concerned with the modularity at the *development phase* of an ontology. Similar to Rector's proposal, our goal is to enable the "TBox" of an ontology to be developed as a set of modules and to later be composed to form one TBox.

However, unlike Rector's approach, we do not restrict a module to taxonomic relations between concepts. Modules are expected to include concepts, relations, and constraints (i.e. a typical TBox). In other words, we do not distinguish modules according to their level of abstraction, or according to the nature of their content. Recall that such a distinction (i.e. "modularization") is achieved by double articulating an ontology into domain and application axiomatizations[60].

The goal of the ontology modularization principle is to enable application axiomatizations to be developed in a modular manner. A module in our approach becomes an application axiomatization where the intended meaning of its vocabulary is defined at the domain axiomatization level, see fig. 4.2.

4.3.1 Modularity criterion

In what follows, we propose a modularity criterion aimed to help ontology builders to achieve *effective decomposition* and to guide them in why/when to release a part of an axiomatization into a separate module. The effectiveness of a decomposition can be seen as the ability to achieve a distributed development of modules and maximize both reusability and maintainability.

> **Subject:** subject-oriented parts should be released into separate modules[61]. For example, when building an axiomatization for university applications, one should separate between the financial aspects (e.g. salary, contract, etc.) and the academic aspects (e.g.

[60] While partitioning an ontology based on the abstraction level of the parts might be called "ontology layering", we use the term "ontology modularization" to refer to modules of the same nature and abstraction level.
[61] This criteria is similar to, the so called "information hiding", in software engineering, [P72].

-D

course, exams, etc.). Encapsulating related axioms (on a certain subject) into one module will not only improve the reusability and maintainability of modules, but also enable the distributed development of modules by different people with a distinct expertise

Purpose: the general-purpose (or maybe called task-oriented) parts of an axiomatization could be released into separate modules. The notion of "general purpose" axiomatization refers to a set of axioms that are expected to be repeatedly used by different kinds of applications. For example, the axiomatization of "payment", "shipping", "person", "address", "invoicing", is often repeated in many e-commerce applications. The reusability of such application axiomatizations is not based necessarily on their ontological foundation or abstraction levels but may be recognized simply from the experience of the creator and from best practices. For example, the wide adoption (i.e. repeatability) of the Dublin Core elements[62] is based mainly on the simplicity of the encoding of descriptions (i.e. metadata) of networked resources.

Specific-purpose parts could also be modularized and released separately. In this way, the application-specificity of other modules will be decreased.

Stability: The parts that are expected to be frequently maintained or replaced could be released in separate modules. This affords other parts more stability and the unstable parts will themselves be easier to maintain and replace.

The criterion suggested above cannot be followed rigidly, as it is based on builders' best practice and expectation of the reuse, maintenance, and distributed development of modules. In chapter 7 we present a case study that illustrates an application of these modularity criterion in the development of a customer complaint ontology.

[62] http://www.dublincore.org (June 2004).

4.3.2 Module composition

To compose modules we define a *composition operator*. All concepts and their relationships (i.e. lexons) and all constraints, across the composed modules, are combined together to form one axiomatization. In other words, the resultant composition is the union of all axioms in the composed modules.

As shall be discussed later, a resultant composition might be *incompatible* in case this composition is not satisfiable, e.g. some of the composed constraints might contradict each other.

Our approach to composition is constrained by the following consistency argument. An ontology builder, *when including a module into another, must expect that all constraints in the included module are inherited by the including module*, i.e. all axioms in the composed modules must be implied in the resultant composition. Formally speaking, the set of possible models for a composition is the intersection of all sets of possible models for all composed modules. In other words, we shall be interested in the set of models that satisfy all of the composed modules.

In fig. 4.3, we illustrate the set of *possible* instances (i.e. possible models) for a concept constrained differently in two modules composed together. Fig. 4.3(a) shows a compatible composition where the set of possible instances for M.**c** is the intersection of the possible instances of M_1.**c** and M_2.**c**. Fig. 4.3(b) shows a case of incompatible composition where the intersection is empty.

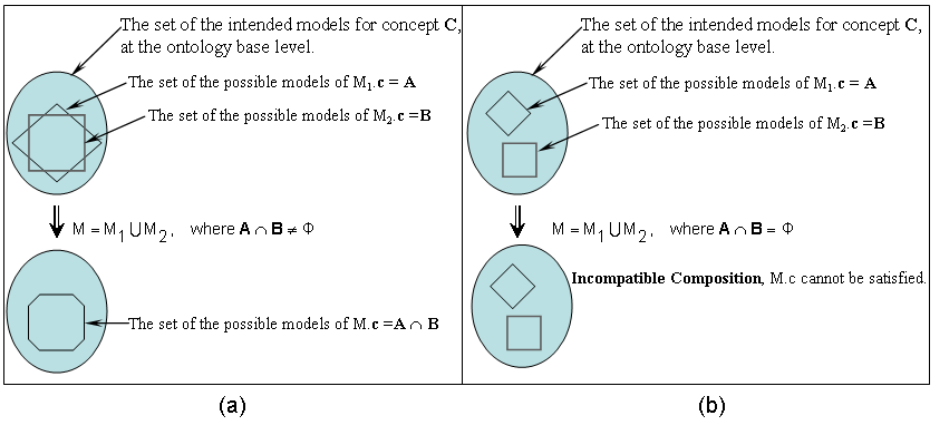

The set of the intended models for concept C, at the ontology base level.
The set of the possible models of M_1.c = A
The set of the possible models of M_2.c =B
$M = M_1 \cup M_2$, where $A \cap B \neq \Phi$
The set of the possible models of M.c =$A \cap B$
(a)
The set of the intended models for concept C, at the ontology base level.
The set of the possible models of M_1.c = A
The set of the possible models of M_2.c =B
$M = M_1 \cup M_2$, where $A \cap B = \Phi$
Incompatible Composition, M.c cannot be satisfied.
(b)

Fig. 4.3. (a) Compatible composition, (b) Incompatible composition.

Notice that our approach to module composition is not intended to integrate or unite the extensions (i.e. ABoxes) of a given set of modules, as several approaches to ontology integration[63] aim to do [SP94] [SK03][BS03]. Our concern is to facilitate ontology builders (at the development phases) with a tool to inherit (or reuse) axiomatizations without “weakening” them. In other words, when including a module into another module (using our composition operator, which we shall formalize in the next section) all axioms defined in the included module should be inherited by (or applied in) the including module.

[63] This might be seen as a designation between composition verses integration of ontological modules.

-D

4.4 Formal framework

In this section, we introduce the formal framework of our approach to module composition. The approach is illustrated, in section 4.5, by developing an algorithm for the automatic composition of modules specified in ORM.

4.4.1 Definition (Module)

A module is an application axiomatization of the form M = <**P**, **Ω**>, where **P** is a non empty set of lexons, i.e. the set of atomic concepts and their relationships; **Ω** is a set of constraints which declares what should necessarily hold in any possible world of M. In other words **Ω** specifies the legal models of M.

4.4.2 Definition (Model, Module satisfiability)

Using the standard notion of an interpretation of a first order theory, an interpretation I of a module M, is a *model*[64] of M iff each sentence of M (i.e. each $\rho \in P$ and each $\omega \in \Omega$) is true for I.

Each module is assumed to be self-consistent, i.e. satisfiable. Module satisfiability demands that each lexon in the module can be satisfied [BHW91]. For each lexon ρ in a given module M, ρ is satisfiable w.r.t. to M if there exists a model *I* of M such that $\rho^I \neq \emptyset$.

Notice that we adopt a strong requirement for satisfiability, as we require each role in the schema to be satisfiable. A weak satisfiability requires only the module itself (as a whole) to be satisfiable [H89][BHW91].

4.4.3 Definition (Composition operator)

Modules are composed by a composition operator, denoted by the symbol '⊕'. Let $M = M_1 \oplus M_2$, we say that M is the composition of M_1 and M_2.

[64] Also called "legal model", see section 3.2.1

M typically is the union of all lexons and constraints in both modules. Let M_1 = <$\mathbf{P}_1$, $\mathbf{\Omega}_1$> and M_2 = <$\mathbf{P}_2$, $\mathbf{\Omega}_2$>, the composition of ($M_1 \oplus M_2$) is formalized as M = < $\mathbf{P}_1 \oplus \mathbf{P}_2$, $\mathbf{\Omega}_1 \oplus \mathbf{\Omega}_2$>.

A composition ($M_1 \oplus M_2$) should *imply* both M_1 and M_2. In other words, for each model that satisfies ($M_1 \oplus M_2$), it should also satisfy each of M_1 and M_2. Let $(M_1)^I$ and $(M_2)^I$ be the set of all possible models of M_1 and M_2 respectively. The set of possible models of $(M_1 \oplus M_2)^I = (M_1)^I \cap (M_2)^I$. A composition is called *incompatible* iff this composition cannot be satisfiable, i.e. there is no model that can satisfy the composition, or each of the composed modules.

In what follow we specify how sets of lexons and sets of constraints can be composed together.

Composing lexons

When composing two sets of lexons ($\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2$), following [M99a], a concept $M_1.\gamma(T)$ in module M_1 and a concept $M_2.\gamma(T)$ in module M_2 are considered exactly the same concept[65] iff they are referred to by the same term T and context γ. Formally, ($M_1.\gamma(T) = M_2.\gamma(T)$) iff ($M_1.\gamma = M_2.\gamma$) and ($M_1.T = M_2.T$). Likewise, two lexons are considered exactly the same ($M_1.<\gamma$: T_1, r, r', $T_2> = M_2.<\gamma$: T_1, r, r', $T_2>$) iff ($M_1.\gamma = M_2.\gamma$), ($M_1.T_1 = M_2.T_1$), ($M_1.r = M_2.r$), ($M_1.r' = M_2.r'$), and ($M_1.T_2 = M_2.T_2$). Indeed, the combination of two sets of lexons can be easily achieved as all lexons share the same definitions of the intended meanings of their vocabularies at the ontology base level[66], see fig. 4.6.

[65] i.e. refer to the same intended models, see section 3.2. and 3.3.
[66] One may notice that another difference between ontology (or schema) integration and composition is in the homogeneity of the integrated/composed modules. In case of integration, all ontologies are expected to be totally heterogeneous. However, in case of composition, modules are expected to have some degree of homogeneity (i.e. evolve within a certain framework). In our approach, modules are assumed to share the same ontology base.

In case that M_1 and M_2 do not share any concept between them (i.e. two disjoint sets of lexons), the composition ($M_1 \oplus M_2$) is considered an incompatible operation[67], as there is no model that can satisfy both M_1 and M_2.

Composing constraints

When composing two sets of constraints, first, all constraints need to be combined together ($\mathbf{\Omega} = \mathbf{\Omega}_1 \oplus \mathbf{\Omega}_2$). Second, a satisfiability reasoning should be performed in order to find out whether the composition ($M = M_1 \cup M_2$) is satisfiable. Finally, an optional step is to perform an implication reasoning to eliminate all implied constraints (also called "entailments") from the composition.

In the first step, the *combination* of all constraints ($\mathbf{\Omega}_1 \oplus \mathbf{\Omega}_2$) should be syntactically valid according to the syntax of the constraint specification language. For example, some constraints need to be syntactically combined into one constraint. *The combination of a set of constraints should imply all of them*. To provide an insight into such combinations, in fig. 4.4, we show the combination of two UML cardinality constraints. Fig. 4.5 illustrates several combinations of ORM constraints. Notice that in case of a constraint contradiction, the composition is terminated and considered an incompatible operation, as in fig. 4.5 (d).

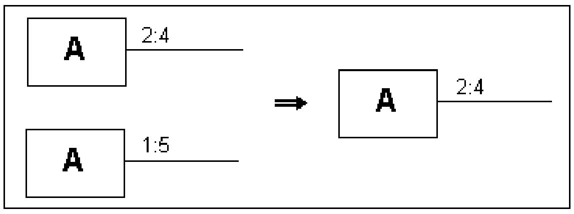


Fig. 4.4. Combining UML constraints.

[67] In some practice cases, we weaken this requirement to allow the composition of disjoint modules. For example, in case one wishes to compose two disjoint modules and later compose them within a third module that results in a joint composition.

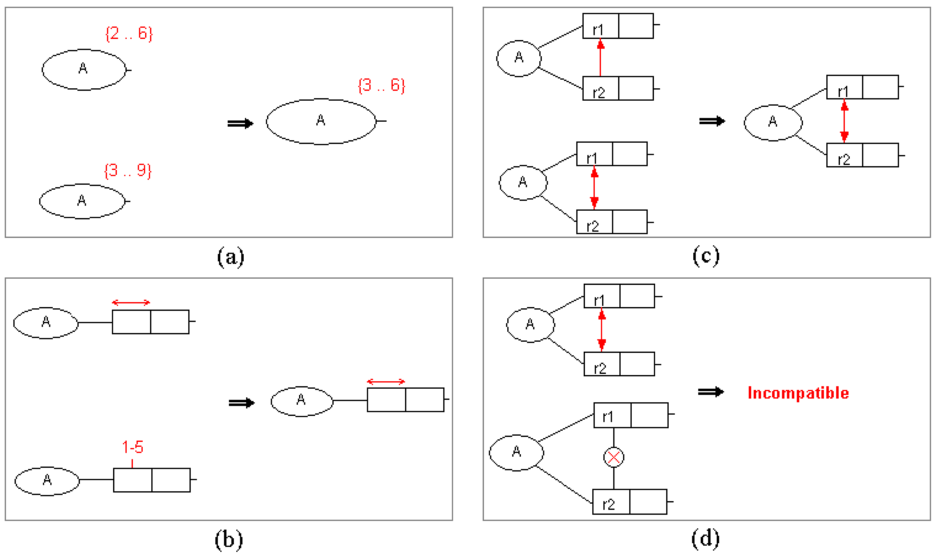


Fig. 4.5. Examples of several combinations of ORM constraints: (a) combination of two value constraints, (b) combination of uniqueness, and frequency, (c) combination of subset and equality, and (d) combinations of equality and exclusion constraints.

The ability to automate this process depends on the complexity of the constraint specification language. Section 4.5 illustrates how all ORM constraints can be combined automatically.

4.4.4 Definition (Modular axiomatization)

A modular axiomatization $M = \{M_1, \ldots, M_n, \oplus\}$ is a set of modules with a composition operator between them, such that $P = (P_1 \oplus \ldots \oplus P_n)$ and $\boldsymbol{\Omega} = (\boldsymbol{\Omega}_1 \oplus \ldots \oplus \boldsymbol{\Omega}_n)$.

Notice that cyclic compositions are null operations, as the repetition of the same proposition has no logical significance. For example, the composition $M = ((M_1 \oplus M) \oplus M_2)$ equals $(M_1 \oplus M_2)$ and the composition $M= ((M_1 \oplus M_2) \oplus (M_2 \oplus M_1))$ also equals $(M_1 \oplus M_2)$.

4.5 Composition of ORM conceptual schemes

As an illustration of our formal framework defined in the previous section, in this section we present an algorithm for automatic composition of modules specified in ORM[68]. An implementation of this algorithm will be presented in chapter 6, as part of our DogmaModeler tool prototype[69].

Each ORM conceptual schema is seen as a module. A concept in the ORM terminology is called an object type, and a relationship is called a predicate. The later consists of a set of roles played by object types. In ORM, a predicate with its associated object types (which we call a lexon), is called a fact type. Other ORM constructs are called constraints, such as Value, Mandatory, Uniqueness, Subset, Equality, Exclusion, Totality, Exclusive and Ring.

We adopt the ORM formalization and syntax as found in [H89][H01], excluding three things. First, although ORM supports n-ary predicates, only binary predicates are considered in our approach. Second, our approach does not support objectification, or the so-called nested fact types in ORM. Finally, our approach does not support the derivation constraints that are not part of the ORM graphical notation[70].

A composition of two modules ($M = M_1 \oplus M_2$) is performed in the following steps: 1) Combine the two sets of fact types ($\mathbf{P} = \mathbf{P}_1 \oplus \mathbf{P}_2$). 2) Combine the two sets of constraints, $\mathbf{\Omega} = \mathbf{\Omega}_1 \oplus \mathbf{\Omega}_2$. 3) Reason to find out whether the composition is satisfiable. Optionally, 4) Reason to eliminate

[68] It is worth to mention that Vermeir [V83] has proposed an approach for modularizing large ORM diagrams based on heuristic procedures. However, this approach is not related to ours, as it is only concerned with how to "view" a one large ORM diagram in different degrees of abstraction or viewpoints. Another similar approach is proposed by Shoval [S85]. Other approaches for viewing large EER diagrams can be found e.g. in [G85] [RS93] [S96]; such approaches are also called clustering methods.

[69] See our motivation on why choosing ORM to illustrate modeling and representing application axiomatization, a long the thesis, in section 5.1.1 and section 3.4.1.

[70] A textual representation of the ORM notation (called ORM markup language) will be presented in chapter 5.

all implied constraints from the composition. The last step is not presented in this thesis as it is quite lengthy. We refer to [H89] for a comprehensive specification of constraint implication in ORM[71].

The composition is considered an incompatible operation (and thus terminated) iff the result cannot be satisfied.

Remark: Although we assume in our formal framework, in section 4.4, that the composition is terminated in case of unsatisfiability, determining whether a composition is satisfiable depends on the *decidability* of the specification language of the composed modules. In case this language is decidable (it has a complete semantic reasoning tableaux), such as OWL, our algorithm can then be called a *complete algorithm*. Otherwise, it is called *incomplete*. In our algorithm of composing ORM schemes, though we reason about the most common unsatisfiability cases, we do not claim this algorithm to be complete, i.e. it is not necessary for the resultant composition to be satisfiable. This is because the general problem of determining consistency for all possible constraint patterns in ORM is undecidable [H97]. A complete semantic tableaux algorithm for deciding the satisfiability of ORM schemes (a research topic by itself) is not a goal of this thesis. We shall build our unsatisfiability cases in this algorithm based on the so-called "ORM formation rules" proposed by Halpin in [H89]. We will also base them on the RIDL-A [DMV], and on the formalization found in [BHW91]. Although these efforts are based on heuristics and do not claim completeness, they cover the most common unsatisfiability cases in practice. As an upcoming effort, we plan to map ORM into the ***DLR*** Description Logic [CDLNR98], which is a powerful and decidable fragment of first order logic. In this way, the satisfiability of ORM schemes can be completely verified.

[71] These steps can also be trivially applied for composing EER and UML schemas.

-D

Step 1: Composing fact types.

In what follows, and for the sake of simplicity, we assume that all object types in all modules have the same context. Two object types of the same terms are considered the same object type. Two fact types of the same terms of the two object types and the two roles are considered the same fact type, i.e. the same lexons. In this way, combing object and fact type across two modules becomes a simple and direct operation, ($\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2$), see figure 4.6.

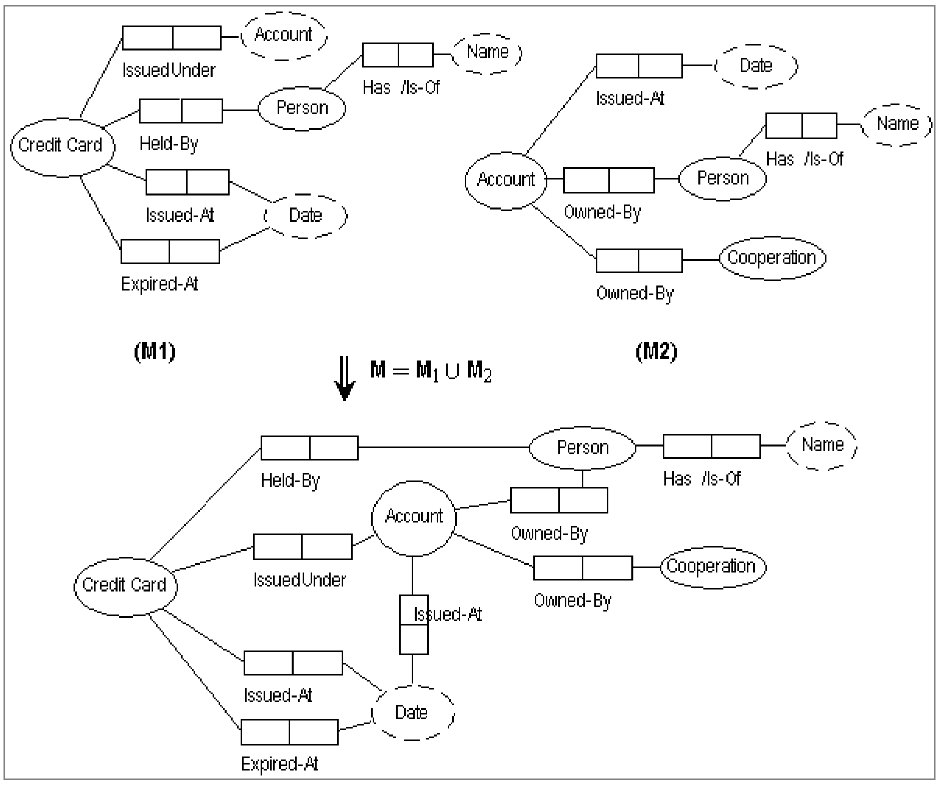


Fig. 4.6. Combining ORM fact types.

Notice that in case an object type is specified as "lexical" in one module and as "non-lexical" in another (e.g. 'Account'), then in the composition, this object type is considered "non-lexical". Lexical object types in ORM are depicted as dotted- ellipsis.

-D

Step 2: Composing constraints.

The goal of this step is to *syntactically* combine the two sets of constraints, i.e. ($\Omega = \Omega_1 \oplus \Omega_2$). Some logical (i.e. satisfiability and implication) validations are also performed in this step, e.g. in case of combining two constraints that contradict or imply each other.

In the following, we show how all ORM constraints can be combined.

Step 2.1: Combining value constraints

The value constraint in ORM indicates the possible values (i.e. instances) for an object type. A value constraint is denoted as a set of values {s1, , sn} depicted near an object type, see fig 4.7. The formalization of the value constraint is $\forall x\ [Ax \equiv x \in \{s_1,\ldots,s_n\}]$. A value in this set can be either a number or a string. The following are some examples: {1,2,3,4}, {2..30}, {1,3,4,9..21,25,30..10}, {'Male', 'Female'}, {1..10,'2','3','a','b'}, etc.

Given two value constraints T.v_1 and T.v_2 on the same object type T, (notice that v_1 and v_2 are two sets of values), their combination is the intersection T.v = $v_1 \cap v_2$, see fig. 4.7(a). If $v_1 \cap v_2$ is empty, then the composition ($M_1 \cup M_2$) is considered as **incompatible operation**, because the value constraints contradict each other and thus the object type cannot be satisfied, see fig. 4.7(b).

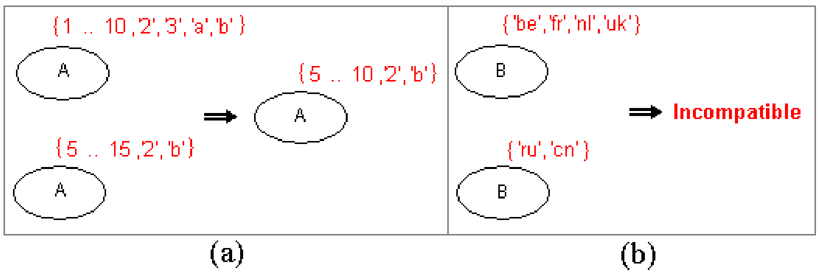


Fig. 4.7. Combining value constraints.

-D

Step 2.2: Combining mandatory constraints

The mandatory constraint in ORM is used to constraint a role (played by an object type) such that each instance of that object type must play this role at least once. See the mandatory constraint in fig. 4.8, which is depicted as a dot on the line connecting the role "IssuedBy" with the object type "Order".

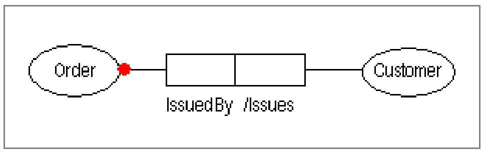


Fig. 4.8. An example of a mandatory constraint.

When composing two modules, all mandatory constraints are included in the composition without any specific combining operation.

Step 2.3: Combining disjunctive mandatory

Disjunctive mandatory constraint is used on a set of two or more roles connected to the same object type. It means that each instance of an object type's population must occur in at least one of the constrained roles. For example, the disjunctive mandatory in fig. 4.9 means that "each account must be owned by a person or a company".

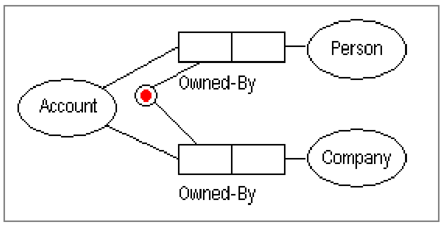


Fig. 4.9. An example of a disjunctive mandatory constraint.

-D

When composing two modules, all disjunctive mandatory constraints are included in the composition without any specific combining operation. See fig. 4.10.

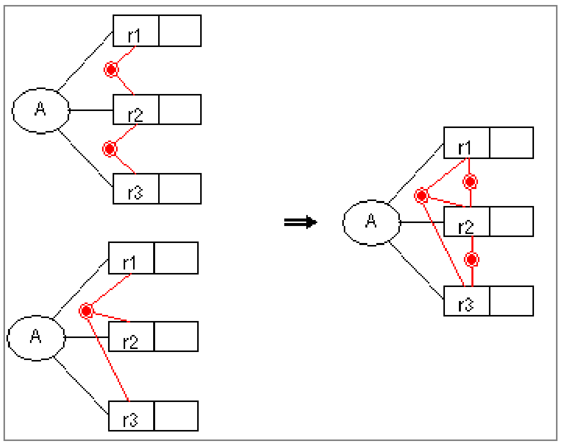


Fig. 4.10. An example of combining disjunctive mandatory constraints.

Step 2.4: Combining uniqueness and frequency constraints

There are three patterns of specifying uniqueness constraints in ORM. An arrow spanning a single role is called "internal" uniqueness, see fig. 4.11(a). It means that "each instance of a book has at most one ISBN", i.e. each occurrence is unique. An arrow spanning the two roles in a predicate is called "predicate" uniqueness, see fig. 4.11(b). It means that "no book can be written by the same author more than once and that no author can write the same book more than once", i.e. a many-to-many constraint[72]. "Inter-predicate" uniqueness constraints, see fig. 4.11(c), apply to roles from different predicates that have a common object type. The roles that participate in a uniqueness constraint uniquely refer to an object type. For example, different values of (author, title, and edition) refer to different

[72] Although this constant has some significance in data modeling, but it is not really a constraint as repetition of a proposition has no logical significance [H89] especially in ontology modeling.

books. In other words, a book can be identified by the values of its author, title, and edition all together.

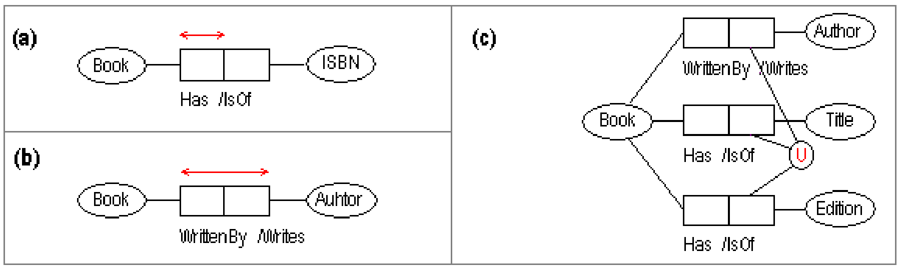


Fig. 4.11. Example of uniqueness constraints.

The frequency constraint (min-max) on a role is used to specify the number of occurrences that this role can be played by its object type. For example, the frequency constraint in fig. 4.12 means, if a car has wheels then it must have at least 3 and at most 4 wheels. Notice that a frequency constraint of maximum 1 is equivalent to an internal uniqueness constraint on this role.

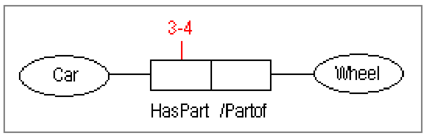


Fig. 4.12. Example of a frequency constraint.

When composing modules, uniqueness and frequency constraints are combined as follows:

1. As internal uniqueness implies predicate uniqueness [H89], the combination of these two constraints is internal uniqueness (see fig. 4.13. (a) and (b)).
2. In case of internal uniqueness and frequency constraints on the same role (see fig. 4.13(c)), the composition of ($M_1 \cup M_2$) is considered an **incompatible operation**, because the two constraints contradict each other [H89], and thus the role cannot be

-D

satisfied. Recall that a frequency of maximum 1 is considered internally uniqueness (see fig. 4.13(d)).

3. In case of two frequency constraints on the same role, FC_1(min-max) and FC_2(min-max), the combination FC(min-max) is calculated as FC.min = Max(FC_1.min, FC_2.min) and FC.max = Min(FC_1.max, FC_2.max), see fig. 4.13(e). In case the FC.min > FC.max, see fig. 4.16(f), then the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**, because the two constraints are in conflict each other, and the role cannot be satisfied.

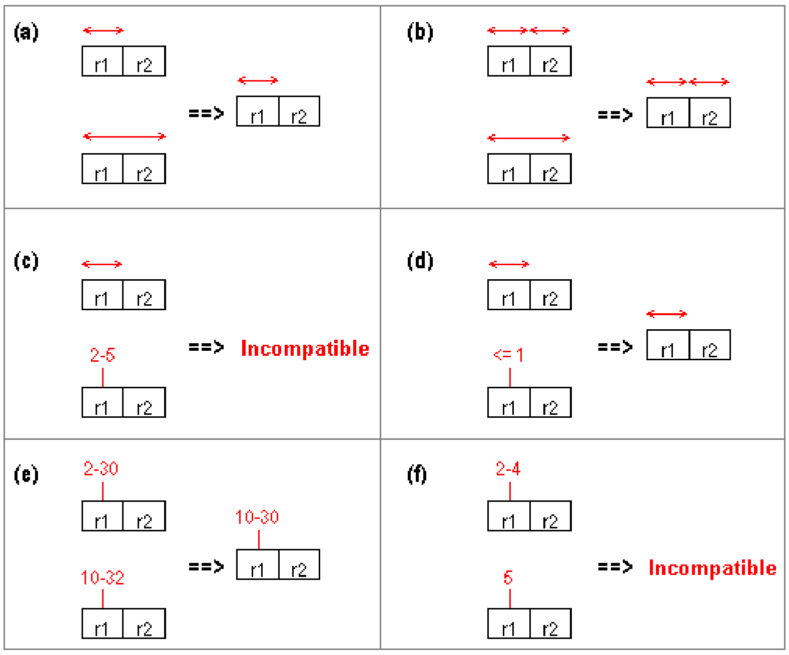


Fig. 4.13. An example of combining uniqueness and frequency constraints.

4. In other cases, all constraints are included in the composition without any specific combining operation. Fig. 4.14 shows an example of combining inter-predicate uniqueness constraints.

-D

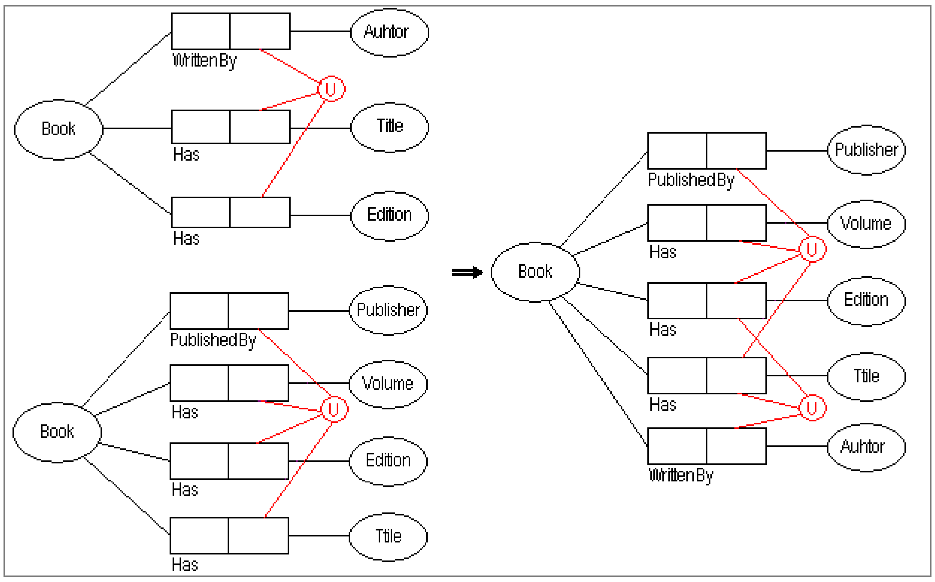


Fig. 4.14. An example of combining inter-predicate uniqueness constraints.

Step 2.5: Combining set-comparison constraints

The set-comparison constraints (subset, equality, and exclusion) are used to restrict the way role(s) is/are populated with respect to other role(s). Fig. 4.15 shows several examples of these constraints. Notice that (only one) set-comparison constraint can be declared either between single roles or between sequences of roles.

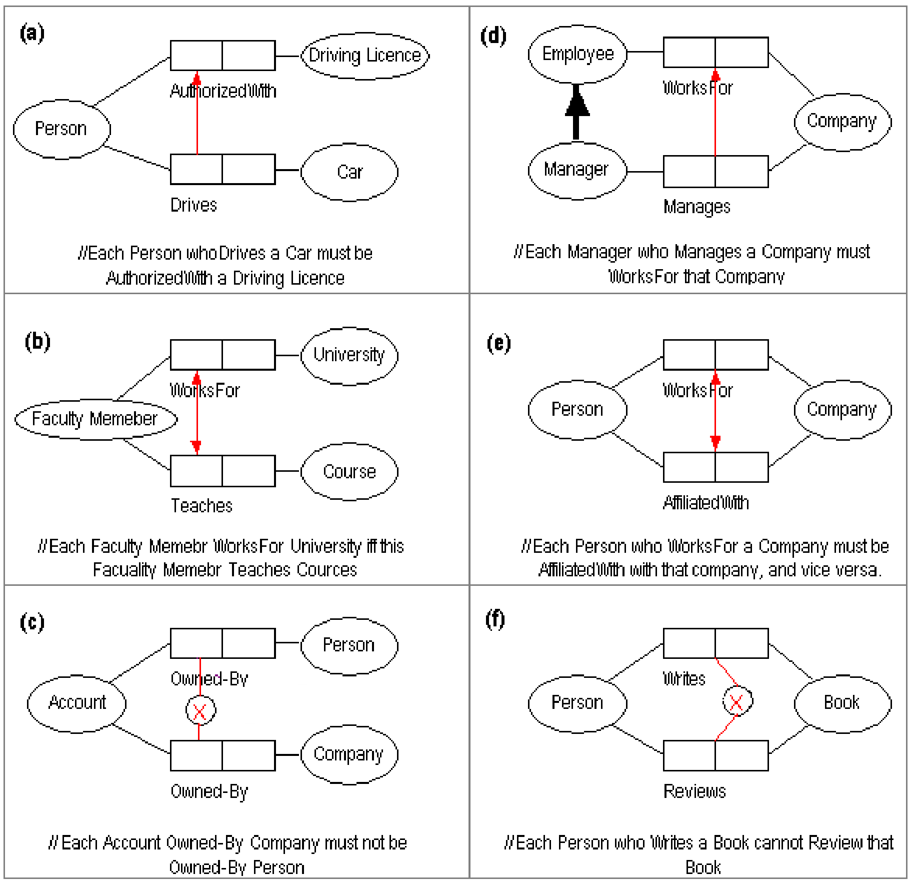

(a)
Person
AuthorizedWith
Driving Licence
Drives
Car
//Each Person who Drives a Car must be AuthorizedWith a Driving Licence
(d)
Employee
WorksFor
Company
Manager
Manages
//Each Manager who Manages a Company must WorksFor that Company
(b)
Faculty Memeber
WorksFor
University
Teaches
Course
//Each Faculty Memebr WorksFor University iff this Facuality Memebr Teaches Cources
(e)
Person
WorksFor
Company
AffiliatedWith
//Each Person who WorksFor a Company must be AffiliatedWith with that company, and vice versa.
(c)
Account
Owned-By
Person
Owned-By
Company
// Each Account Owned-By Company must not be Owned-By Person
(f)
Person
Writes
Book
Reviews
//Each Person who Writes a Book cannot Review that Book

Fig. 4.15. Examples of set-comparison constraints.

Combining set-comparison constraints across two modules is performed in the following steps:

1. Each exclusion constraint that spans more than two singles or sequences of roles (called "multiple" exclusion) is converted into pairs of exclusions[73], such in Fig. 4.16.

[73] This conversion is temporary for reasoning purposes, so it will not appear in the final result of the composition. Notice that "a single exclusion constraint a cross n roles replaces $n(n-1)/2$ separate exclusion constraints between two roles" [H01].

-D

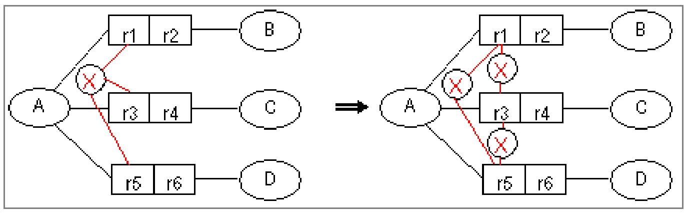


Fig. 4.16. Converting multiple exclusions into pairs of exclusions.

2. When combining a subset (or equality) in one module and an exclusion in another, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**, because the two constraints contradict each other, and so both roles cannot be satisfied. See fig. 4.17.

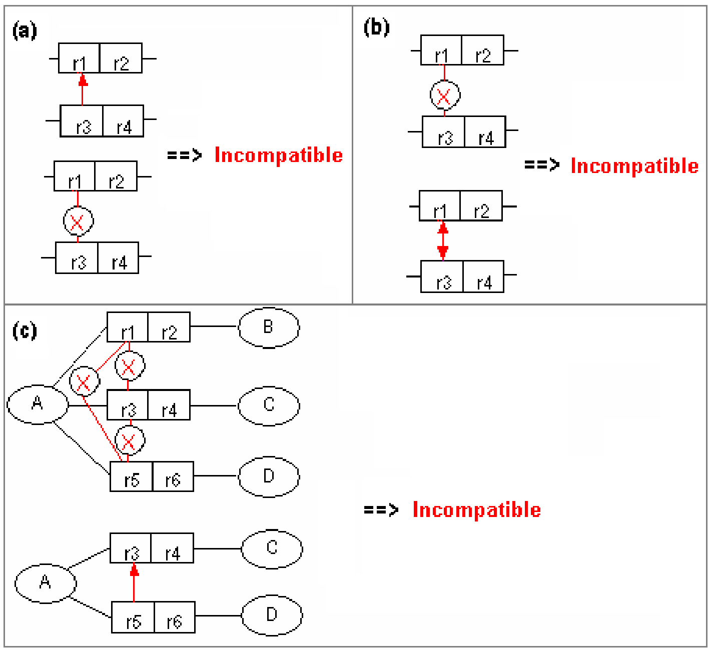


Fig. 4.17. Combining subset (or equality) with exclusion.

3. As equality implies subset (but not vice versa) [H89], when combining a subset in one module and equality in another module, or when combining two subset constraints that are opposite to each other, the combination is always equality. See Fig. 4.18.

-D

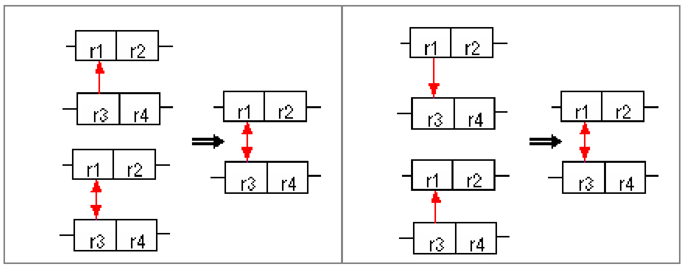


Fig. 4.18. Combining subset and equality constraints.

Step 2.6: Combining subtype constraints (total, exclusive)

Total and exclusive constraints can only be declared on a set of subtypes sharing the same supertype, see fig 4.19.

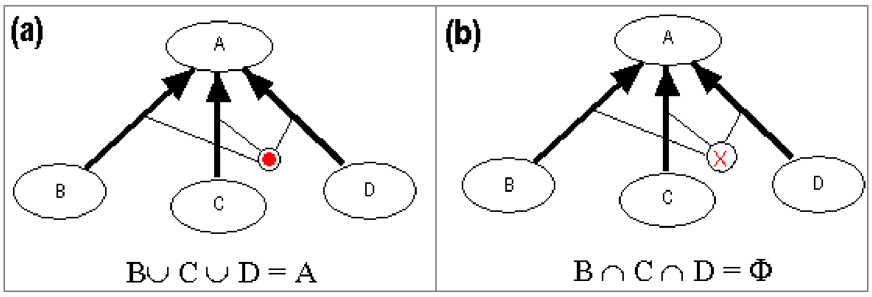


Fig. 4.19. Examples of subtype constraints: (a) total, (b) exclusive.

When composing two modules, all subtype constraints are included in the composition without any specific combining operation, see fig. 4.20.

-D

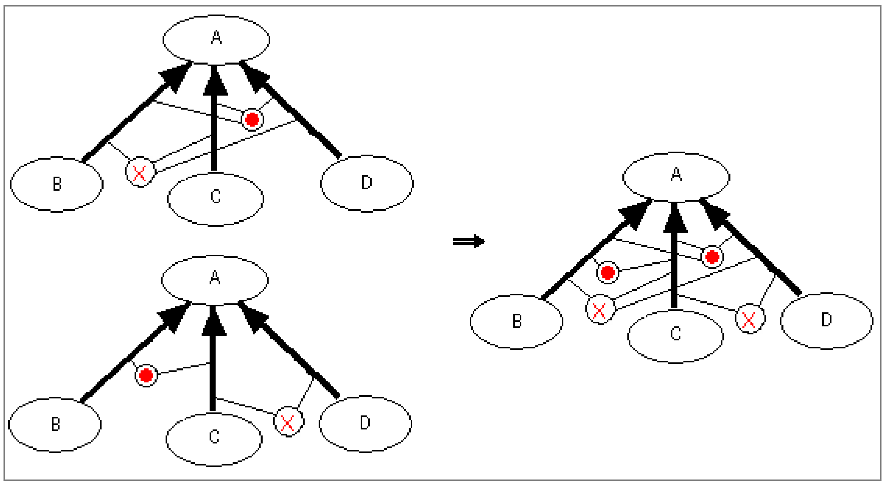


Fig. 4.20. Combining subtype constraints.

Notice that constraint implications, such as the exclusive constraint between (C, D) that is implied by the exclusive constraint between (B, C, and D), are not resolved in this step.

Step 2.7: Combining ring constraints

ORM allows ring constraints to be applied to a pair of roles that are connected directly to the same object type in a fact type, or indirectly via supertypes. Six types of ring constraints are supported by ORM: antisymmetric (ans), asymmetric (as), acyclic (ac), irreflexive (ir), intransitive (it), and symmetric (sym) [H01][H99]. Fig. 4.21 shows several examples of these constraints. Combinations of ring constraints on the same pair of roles are also allowed, such as in fig. 4.21 (a) and (e).

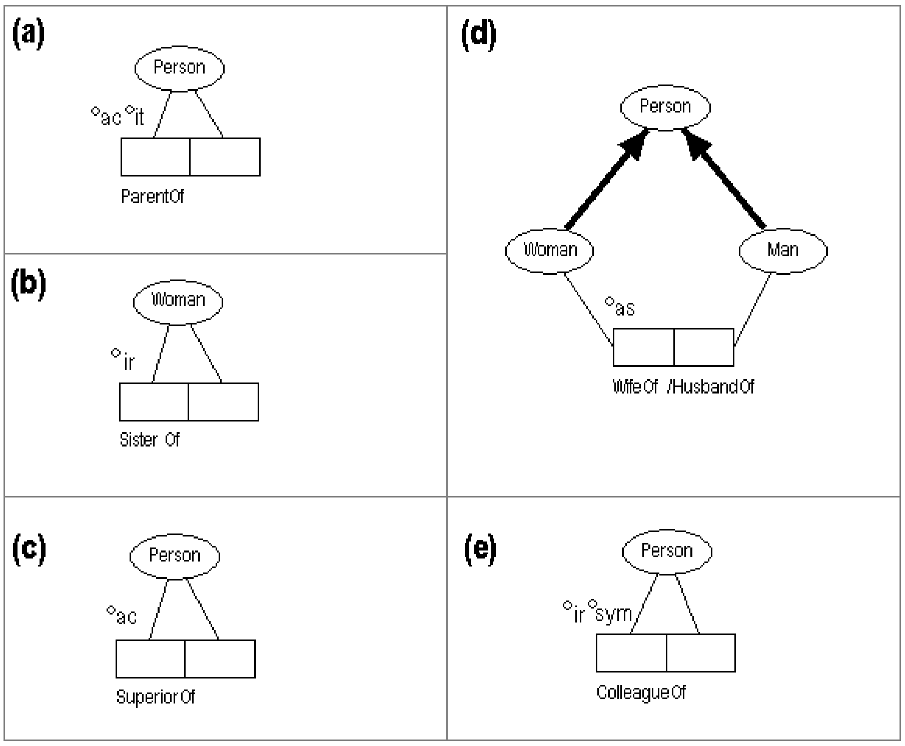


Fig. 4.21. Examples of ring constraints.

The relationships between the six ring constraints are formalized by [H01] using the Eular diagram as in fig. 4.22. This formalization helps one to visualize the implication and incompatibility between the constraints. For example, one can see that acyclic implies reflexivity, intransitivity implies reflexivity, the combination between antiasymmetric and reflexivity is exactly asymmetric, and acyclic and symmetric are incompatible.

-D

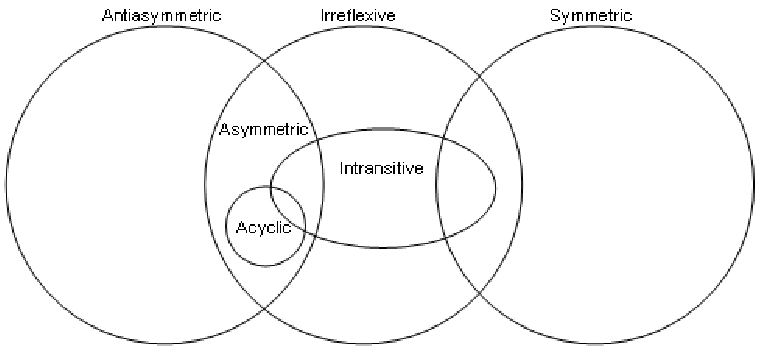
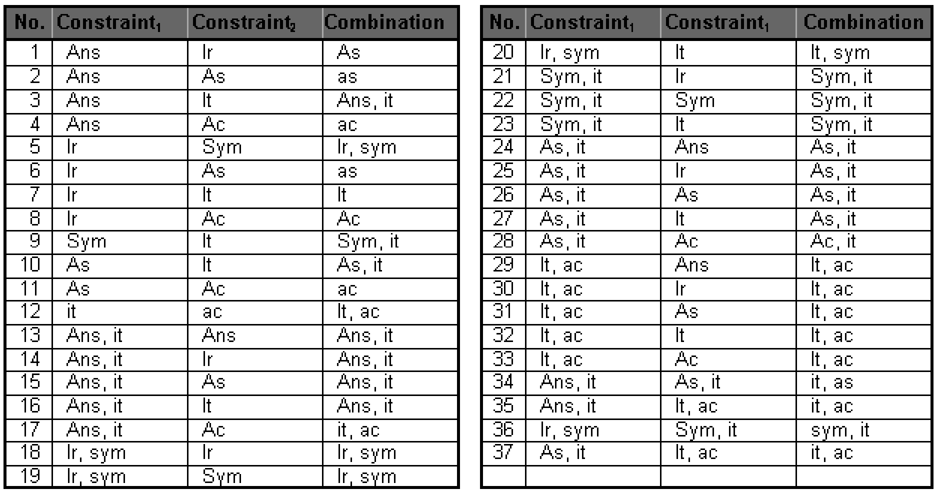


Fig. 4.22. Relationships between ring constraints [H01].

When composing two modules, ring constraints are combined based on the formalization in fig. 4.22. Any combination of ring constraints should be compatible, i.e. there is an intersection between their zones in the Eular diagram. Otherwise, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**, because the combined rings constraints conflict each other, and thus the role cannot be satisfied.

Based on the Eular diagram, in table 4.1 we derive all possible *compatible* combinations of the six ring constraints. Combinations that do not appear in the table are incompatible, such as (ans) and (ac), (Sym, it) and (Ans), (Sym, it) and (It, ac), or (Ans, it) and (Ir, sym), etc.

-D



No.	Constraint$_1$	Constraint$_2$	Combination
1	Ans	Ir	As
2	Ans	As	as
3	Ans	It	Ans, it
4	Ans	Ac	ac
5	Ir	Sym	Ir, sym
6	Ir	As	as
7	Ir	It	It
8	Ir	Ac	Ac
9	Sym	It	Sym, it
10	As	It	As, it
11	As	Ac	ac
12	it	ac	It, ac
13	Ans, it	Ans	Ans, it
14	Ans, it	Ir	Ans, it
15	Ans, it	As	Ans, it
16	Ans, it	It	Ans, it
17	Ans, it	Ac	it, ac
18	Ir, sym	Ir	Ir, sym
19	Ir, sym	Sym	Ir, sym
20	Ir, sym	It	It, sym
21	Sym, it	Ir	Sym, it
22	Sym, it	Sym	Sym, it
23	Sym, it	It	Sym, it
24	As, it	Ans	As, it
25	As, it	Ir	As, it
26	As, it	As	As, it
27	As, it	It	As, it
28	As, it	Ac	Ac, it
29	It, ac	Ans	It, ac
30	It, ac	Ir	It, ac
31	It, ac	As	It, ac
32	It, ac	It	It, ac
33	It, ac	Ac	It, ac
34	Ans, it	As, it	it, as
35	Ans, it	It, ac	it, ac
36	Ir, sym	Sym, it	sym, it
37	As, it	It, ac	it, ac

Table 4.1. All possible combatable combinations or ring constraints.

Step 3: Reasoning about the satisfiability of ORM modules[74]

Some unsatisfiability cases were detected in the previous step, in particular those that emerged when two or more constraints were combined. In this step, we reason about other cases that may emerge between different constraints in the composition.

As we noted earlier, as the completeness of our algorithm depends on the decidability of the modules' language, it is not necessary for the resultant composition in this algorithm to be completely satisfiable. This is because the general problem of determining consistency for all possible constraint patterns in ORM is un-decidable [H97]. See our discussion on this issue in the previous section.

In what follows, we present six cases of constraint patterns that lead to unsatisfiability. These patterns are compiled from [H89][H03][BHW91] [DMV] and refined to suit our reasoning about module satisfiability. Although we do not claim completeness, these patterns - in addition to the

[74] Later, this section was revised in and extended, see [JS06] and [JH08].

unsatisfiability cases that we have shown in the previous step - cover the most common unsatisfiability cases in practice.

Pattern 1 (Top common supertype)

In this pattern, subtypes that do not have a top common supertype are detected. In ORM, all object types are assumed by definition to be mutually exclusive, except those that are subtypes. Thus, if a subtype has more than one supertype, these supertypes must share a top supertype; otherwise, the subtype cannot be satisfied. In fig. 4.23, the object type C cannot be satisfied because its supertypes A and B do not share a common supertype, i.e. A and B are mutually exclusive.

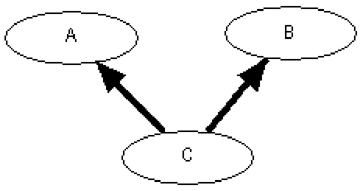


Fig. 4.23. Subtype without a top common supertype.

Formally, for each subtype T, let T.DirectSupers be the set of all direct supertypes of T. Let T.DirectSupers$_i$.Supers be the set of all possible supertypes of T.DirectSupers$_i$. If $(T.DirectSupers_1.supers \cap ... \cap T.DirectSupers_n.supers) = \Phi$, then the object type T cannot be satisfied. In this case, the composition ($M_1 \cup M_2$) is considered an **incompatible operation.**

For implementation purposes, the following algorithm is another presentation[75] of the above formalisms.

Algorithm:

[75] We use the object-oriented data structure to write our algorithms for the sake of brevity, and for the simplicity of implementation in modern programming languages. The algorithms are written in a simple JAVA-alike pseudo language. We present the implementation of the six patterns in DogmaModeler in section 6.4.

```
For each subtype T[x] {
 Let T[x].DirectSupers = the set of all direct supertypes of T[x].
 n = T[x].DirectSupers.size
 If ( n > 1)  {
   For (i = 1 to i=n)    {
     Let T[x].DirectSupers[i].Supers = the set of all possible supertypes
                                                     of T[x].DirectSupers[i]   }
   // if the intersection of all T[x].DirectSupers[i].supers is not empty,
       then the composition is not satisfiable.
   if (Intersection(T[x].DirectSupers[1].supers,      T[x].DirectSupers[n].supers))
       is empty  {
        Composition.Satisfiability = false
        Message= ("The subtype T[x].DirectSupers[i] cannot
             be satisfied as its supertypes do not have a top common supertype.")
     }}
}
```

Pattern 2 (Exclusive constraint between types)

In this pattern, subtypes of mutually exclusive supertypes (caused by an exclusive constraint) are detected. Fig. 4.24 shows a case where D cannot be satisfied because its supertypes are mutually exclusive. The set of instances of D is the intersection of the instances of B and C, which is an empty set according to the exclusive constraint between B and C.

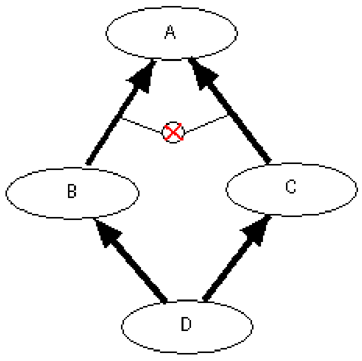


Fig. 4.24. Subtype with exclusive supertypes.

Formally, for each exclusive constraint between a set of object types $T = \{T_1, \ldots T_n\}$, let T_i.Subs be the set of all possible subtypes of the object

-D

type T_i, and T_j.Subs be the set of all possible subtypes of the object type T_j, where $i \neq j$, the set (T_i.Subs $\cap$ T_j.Subs) must be empty. Otherwise members in this set are not satisfiable; and hence, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**.

For implementation purposes, the following algorithm is another presentation of the above formalisms.

Algorithm:

```
For each exclusive constraint Exv[x] {
 Let Exv[x].T = the set of the object types participating in Exv[x].
 //For each pair of object types participating in the exclusion constraint:
 For (i = 1 to i = Exv[x].T.size) {
    For (j = 1 to j = Exv[x].T.size) {
       If (i not equal j) {
          Let Exv[x].T[i].Subs = the set of subtypes of the object type Exv[x].T[i].
          Let Exv[x].T[j].Subs = the set of subtypes of the object type Exv[x].T[j].
          S = IntersectionOf(Exv[x].T[i].Subs, Exv[x].T[j].Subs)
          If (S is not empty) {
             Composition.Satisfiability = false
             Message = ("all subtypes in <S> cannot be
                        instantiated because of <Exv[x]>") }}}}
}
```

Pattern 3 (Exclusion-Mandatory)

In this pattern, contradictions between exclusion and mandatory constraints are detected. In Fig. 4.25, we show three examples of unsatisfiable schemes.

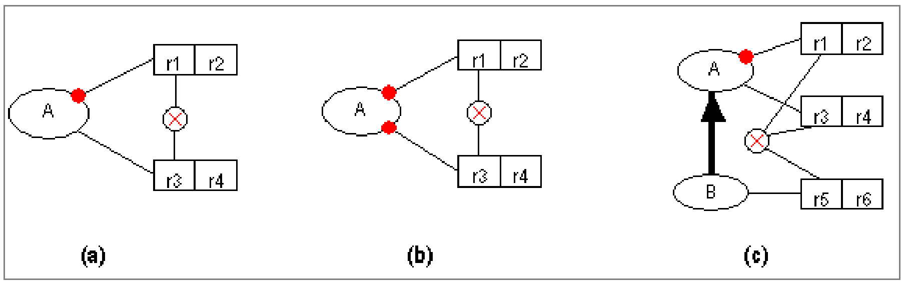


Fig. 4.25. Unsatisfiable schemes because of the mandatory and exclusion conflicts.

In the first case (a), the role r3 will never be played. The mandatory and exclusion constraints restrict that each instance of A must play r1 and the instance that plays r1 cannot play r3. In the second case (b), both r1 and r3 will never be played. According to the two mandatory constraints, each instance of A must play both r1 and r3. At the same time, according to the exclusion constraints, an instance of A cannot play r1 and r3 together. Likewise, in the third case (c), r3 and r5 will never be played. As B is a subtype of A, instances of B inherit all roles and constraints from A. For example, if an instance of B plays r5, then this instance - which is also an instance of A - cannot play r1 or r3. However, according to the mandatory constraint, each instances of A must play r1 and according to the exclusion constrain, it cannot play r1, r3 and r5 all at the same time.

In general, a contradiction occurs if an object type that plays a mandatory role participates in an exclusion constraint with other roles played by this object type or one of its subtypes.

Formally, for each exclusion constraint between a set of single roles R, let $R_i.T$ be the object type that plays the role R_i, $R_i \in R$. For each (R_i, R_j), where $i \neq j$ and R_i *is mandatory*, if $R_i.T = R_j.T$ or $R_j.T \in R_i.T.Subs$ -where $R_i.T.Subs$ is the set of all subtypes of the object type $R_i.T$ - then some roles in R cannot be populated. Hence, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**.

-D

For implementation purposes, the following two alternative algorithms are another presentation of the above formalism.

Algorithm:

```
For each exclusion constraint Exs[x] between a set of single roles {
 Let Exs[x].roles = the set of all roles participating in Exs[x].
 For (i=1 to Exs[x].roles.size)
   If (Exs[x].roles[i].Mandatory = true) {
     For (j=1 to Exs[x].roles.size) {
       If (I not equal j){
        Let Exs[x].roles[i].T = the object type that plays the role Exs[x].roles[i]
        Let Exs[x].roles[j].T = the object type that plays the role Exs[x].roles[j]
        Let Exs[x].roles[i].T.Subs = the set of all subtypes of Exs[x].roles[i].T
        If (Exs[x].roles[i].T = Exs[x].roles[j].T) OR
                                  In(Exs[x].roles[j].T, Exs[x].roles[i].T.Subs ) {
            Composition.Satisfiability = false
            Message = ("There are some roles in <Exs[x].roles> that cannot
              be instantiated because of the <Exv[x]>")}}}}}
```

An alternative but more compact algorithm can be:

```
For each exclusion constraint Exs[x] between a set of single roles {
 Let Exs[x].roles = the set of all roles participating in Exs[x].
 Let MandRoles = the set of all mandatory roles from Exs[x].roles.
 If (MandRoles  is not empty)
    For (i=1 to ManRoles.size)
      For (j=1 to Exs[x].roles.size)
        Let MandRoles[i].T = the object type that plays the role MandRoles[i]
        Let Exs[x].roles[j].T = the object type that plays the role Exs[x].roles[j]
        Let Exs[x].roles[j].T.Subs = the set of all subtypes of Exs[x].roles[j].T
        If Not In(MandRoles[i].T, Exs[x].roles[j].T.Subs)
            Composition.Satisfiability = false
            Composition.Satisfiability.reason= ("There are some roles in
              <Exs[x].roles> that cannot be populated because of the <Exv[x]>")}}}}
}
```

Pattern 4: (Frequency-Value)

In this pattern, contradictions between value and frequency constraints are detected.

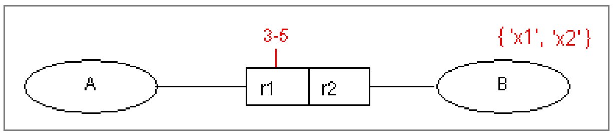


Fig. 4.26. Contradiction between value and frequency constraints.

In fig. 4.26, the role r1 cannot be populated. If the frequency constraint (3-5) on r1 is satisfied, each instance of A must play r1 at least three times, and thus three different instances of B are required. However, there are only two possible instances of B, which are declared by the value constraint {‘x1’, ‘x2’}.

For each fact type $(A\, r\, B)$*, let c be the number of the possible values of* B *that can be calculated from its value constrain, and let* $(n - m)$ *be a frequency constraint on the role* r, c *must be equal or more than* n. Otherwise, the role r cannot be satisfied, as the value and the frequency constraints contradict each other. Hence, ($M_1 \oplus M_2$) is considered an **incompatible operation**.

For implementation purposes, the following algorithm is another presentation of the above formalisms.

Algorithm:

```
For each frequency constraint F[x] {
   Let F[x].min = the lower bound of the frequency constraint F[x].
   Let T = the object type that is played by the role holding F[x].
   Let T.Values = the value constraint on T.
   // if there is no value constraint on T, then T.Values = null
   If (T.Values is not null) and (T.Values.size < F[x].min) {
       Composition.Satisfiability = false.
       Message =("the role <T.r> cannot be instantiated because the
           <F[x]> and the <T.Values> contradict each other"). }
}
```

Pattern 5 (Value-Exclusion)

-D

Contradictions between value and exclusion constraints are detected in this pattern. Fig. 4.27 shows a contradiction between the exclusion and the value constraints. This contradiction implies that one of the roles that is connected to A cannot be populated. According to the exclusion constraint, there should be at least three different values of A to play r1, r3 and r5. However, according to the value constraint, there are only two possible values of A.

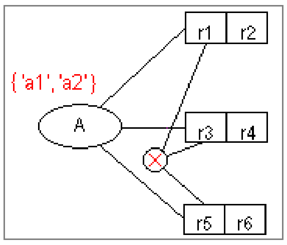


Fig. 4.27. Contradiction between value and exclusion constraints.

For each exclusion constrain, let $R=\{R_1,\ldots,R_n\}$ be the set of roles participating in this constraint, and let *n* be the number of the roles in R. Let T be the object type that plays all roles in R. Let C be the number of possible values of T, according to value constraint. C *must always be more than or equal n*. Otherwise, some roles in R cannot be satisfied, and hence, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**.

For implementation purposes, the following algorithm is another presentation of the above formalisms.

Algorithm:

```
For each exclusion constraint Exs[x] between a set of single roles {
  Let Exs[x].Roles = the set of roles participating in the exclusion Exs[x].
  Let O = the object type that plays all roles in Exs[x].Roles.
  Let O.Values = the value constraint on O.
  // if there is no value constraint on O, then O.Values = null
  If (O.Values is not null) and (O.Values.size < Exs[x].Roels.size) {
       Composition.Satisfiability = false.
```

-D

```
    Message =("Some roles in <Exs[x].Roles> cannot be instantiated because
        the <Exs[x]> and the <O.Values> contradict each other").}
}
```

Pattern 6 (Set-comparison constraints)

In this pattern, contradictions between exclusion, subset, and equality constraints are detected. Fig. 4.28 shows a contradiction between the exclusion and the subset constraints. This contradiction implies that both predicates cannot be populated.

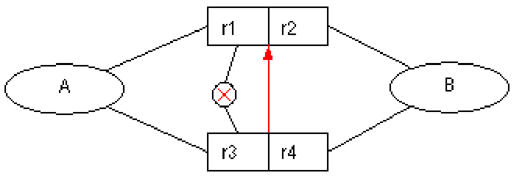


Fig. 4.28. A non fact type populatable schema.

The exclusion constraint between the two roles r1 and r3 means that their populations should be distinct. However, in order to satisfy the subset constraint between (r1, r2) and (r3, r4), the populations of r1 and r3 should not be distinct. In other words, the exclusion constraint between r1 and r3 implies an exclusion constraint between (r1, r2) and (r3, r4) [H89], which contradicts any subset or equality constraint between both predicates.

Fig. 4.29 shows the implications for each set-comparison constraint that might be declared between parts of role sequences. These implications are taken into account when reasoning for contradictions between the three set-comparison constraints.

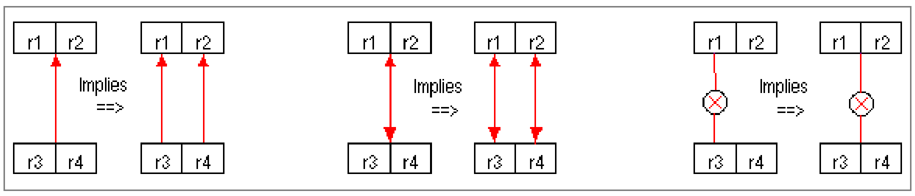


Fig. 4.29. Main set-comparison implications [H01].

-D

In addition, an equality constraint is equivalent to two subset constraints. Hence, we refer to a subset or an equality constraint as a SetPath.

For each exclusion constraint between A and B: If A and B are two predicates, there should not be any (direct or implied) SetPath between these predicates; If A and B are single roles, there should not be any (direct or implied) SetPath between both roles or between the predicates that include these roles.

Otherwise, the two predicates cannot be populated, as the two constraints contradict each other. In this case, the composition of ($M_1 \oplus M_2$) is considered an **incompatible operation**.

Algorithm:

```
For each exclusion constraint Exs[x] {
 If (Exs[x] between predicates) {
   Let Exs[x].predicates = the set of all predicates participating in Exs[x].
   \\ For each pair of predicates participating in the exclusion
   For (i = 1 to i = Exs[x].predicates.size) {
     For (j = 1 to j = Exs[x].predicates.size) {
       If (i not equal j) {
         Sp = GetSetPathsBetween(Exs[x].Predicates[i], Exs[x].Predicates[j])
         // Sp is the set of all subset or equality constraints that specify or imply a
         // SetPath between the current tuple of predicates.
         If (Sp is not empty) {
           Composition.Satisfiability = false.
           Message = ("the exclusion constraint <Exs[x]> contradicts some subset
                     and/or equality constraints on the predicates in <Sp>").}}}}}
 Else { // then the Exs[x] is between roles
   Let Exs[x].roles = the set of all roles that participate in Exs[x].
   \\ For each pair of roles participating in the exclusion constraint
   For (i = 1 to i = Exs[x].roles.size) {
     For (j = 1 to j = Exs[x].roles.size) {
       If (i not equal j) {
         Sr = GetSetPathsBetween(Exs[x].roles[i], Exs[x].roles[j])
         // Sr is the set of all subset or equality constraints that specify or imply a
         // SetPath between the current tuple of roles.
         Sp = GetSetPathsBetween(Exs[x].Predicates[i], Exs[x].Predicates[j])
```

-D

```
        // Sp is the set of all subset or equality constraints that specify or imply a
        // SetPath between the predicates of the current tuple of roles.
        If (Sr is not empty) OR (Sp is not empty) {
          Composition.Satisfiability = false.
          Message = ("the exclusion constraint <Exs[x]> contradicts some Subset
               and/or equality constraints on the predicates in Sp"). }}}}}}
}
```

4.6 Discussion and conclusions

In this chapter, we have presented the ontology modularization principle. We have shown how application axiomatizations can be developed as modules and later composed to form one modular axiomatization. In the following paragraphs, we summarize the main advantages of the ontology modularization principle:

- *Modules are easy to reuse in other kinds of applications.* In addition to our contribution towards the reusability of domain axiomatizations (which can be achieved by the double-articulation principle), the reusability of application axiomatizations can also be improved by modularizing it into a set of compose-able modules. The two engineering principles indeed complement each other. By the double-articulation principle, the ontology reusability is improved by separating between domain and application axiomatizations based on the abstraction level of axioms. Correspondingly, the modularization principle contributes to ontology reusability by enabling parts of application axiomatizations to be isolated and reused among other application-kinds. **Hence, we fulfill the R1 engineering requirement**: Ontologies should be engineered in a way that allows the isolation and identification of the reusable parts of an ontology.
- *Enable distributed development of modules over different locations, expertise, and stakeholders*. The double-articulation and modularization principles complement each other also in the distributed development of ontologies. While the double articulation principle enables (domain experts, lexicographers, knowledge engineers, etc.) to contribute to the development of domain axiomatizations, the modularization principle enables the application axiomatization development to be distributed among

different application-oriented expertise, stakeholders, etc. As we have shown in our example in section 4.1.1, while the "Payment" module might be developed and released by a company specialized in online payment services, the "BookOrder" module can be developed and released by bookstore companies. Such modules can be composed later to form one book-shopping axiomatization. **Hence, we claim to fulfill the R4 engineering requirement**: The ontology representation model should be capable of distributed and collaborative development.

- *Modules are easier to build, maintain, and replace*. This is because the internal couplings (e.g. the number of relationships between concepts) in small modules are fewer than the internal couplings in large axiomatizations. The development and maintenance of small modules enable ontology builders a better focus and easy understanding than large and multi-domain axiomatizations[76]. The modularity of an axiomatization also enables ontology users and maintainers to interchange some parts with others that are for example, more relevant, reliable or accurate. In short, the modularization principle indeed enables the evolution life cycle of axiomatizations to be more efficient. **Hence, modularization assists in fulfilling the R5 and R6 engineering requirement.** Ontologies should be engineered in a way that enables smooth and efficient evolution (R5). Ontologies should be engineered in a way that allows easy replacement of the axiomatization of ontology parts (R6).

[76] The reader may noticed that our contribution towards ontology maintainability is not concerned with the consequences of ontology evolution (on running applications), as versioning mechanisms (cf. [Hj01], [KKOF02], [MMS03])) are intended to resolve. Our main concern is on how to make the ontology evolution process itself easy and more efficient. Nevertheless, it would be easier for versioning mechanisms to keep track of changes in modules than changes in the whole ontology. As we have discussed earlier, unsteady part of an ontology can be realized into a separate module, which steadies the other modules.

- *Enable effective management and browsing of modules.* Modules are easier to store, retrieve, search, index, and master than large and multi-domain axiomatizations. In chapter 5 and 6, we show a prototype of a library of modular axiomatizations, where modules are annotated and indexed using Dublin-Core metadata. In addition, we will show how axiomatizations can be effectively browsed and viewed as modules.

This chapter concludes our discussion of the methodological principles of our thesis. Next, we proceed to present the *implementation* .

Part III

Implementation

Implementation (WordNet 1.7.1):

[1]-The act of accomplishing some aim or executing some order. E.g. "the agency was created for the implementation of the policy"

[2]-The act of implementing (providing a practical means for accomplishing something); carrying into effect."

-(http://wordnet.princeton.edu)

In this part, we present the implementation part of the thesis. The next chapter defines a conceptual markup language of the ORM graphical notation. In chapter 6, we present an ontology engineering tool called DogmaModeler. In chapter 7, we present our experience and achievements on applying our methodological principles and tool in building a -medium size- costumer complaint ontology.

-D

Chapter 5

ORM Markup Language

"... Quite a number of knowledge representation techniques are supported by some kind of graphical formalism, usually called a "semantic network" of sorts.....Semantic nets allow to construct an explicit connection between on the one hand "Al-style" knowledge representation and on the other hand "classical" database design. ...".

-(R. Meersman, [M86])

In this chapter, we define a conceptual markup language (ORM-ML) for the ORM graphical notation. In section 5.1 we provide a brief introduction and discuss our motives for constructing the ORM markup language before we present the language itself in section 5.2. To end, section 5.3 draws some conclusions and summarizes the main advantages of ORM-ML.

5.1 Introduction and motivation[77]

In this chapter, we define a conceptual markup language for the ORM graphical notation. This language will be used in our DogmaModeler tool prototype (in chapter 6) for representing application axiomatizations.

The ORM markup language presented in this chapter is an intensively improved version (Version 2.0) of the language that we have published in [DJM02a][DJM02b][JDM03].

Although application axiomatizations might be specified in different specification languages (see section 3.4), we have chosen to *illustrate* our approach using ORM.

Indeed, successful conceptual data modeling approaches, such as ORM or EER, became well known because of their methodological guidance in building conceptual models of information systems. They are semantically rich disciplines and support quality checks at a high level of abstraction [V82] and they provide modeling constructs like integrity, taxonomy, and derivation rules [H01] [F02]. Merely, conceptual data schemes -also called semantic data models - were developed to capture the meaning of an application domain as perceived by its developers [WSW99] [M99a]. This meaning is being represented in diagram formats (which are proprietary and therefore are limited to use inside specific CASE tools), and typically used in an *off-time mode*, i.e. used during the design phases. Nowadays, the Internet and the open connectivity environments create a strong demand for sharing and exchanging not only data but also data semantics. By defining a conceptual markup language (ORM-ML) that allows for the representation of ORM conceptual diagrams in an open, textual syntax, we enable ORM schemes to be shared, exchanged, and processed at run-time.

[77] Later, this section was revised and extended, see [J07a].

5.1.1 Why ORM

ORM (Object-Role Modeling) [H01] is a conceptual modeling approach that was developed in the early 70's. It is a successor of the NIAM (Natural-language Information Analysis Method) [VB82]. Based on ORM, several conceptual modeling tools exist, such as Microsoft's VisioModeler™ and the older InfoModeler. This has the functionality of modeling a certain Universe of Discourse (UoD) in ORM while supporting the automatic generation of a consistent and normalized relational database schema.

ORM schemas can be translated into pseudo natural language statements. The graphical representation and the translation into pseudo natural language make it a lot easier, also for non-computer scientists, to create, check and adapt the knowledge about the UoD needed in an information system.

The ORM conceptual schema methodology is fairly comprehensive in its treatment of many "practical" or "standard" business rules and constraint types. Its detailed formal description, (we shall take ours from [H01][H89]) makes it an interesting candidate to non-trivially illustrate our XML based ORM-markup language as an exchange protocol for representing ORM conceptual models (seen as application axiomatizations).

Of course, similar to ORM-ML, a markup language could be defined for any other conceptual modeling method. We have chosen ORM *to illustrate the adoption of conceptual data modeling methods for ontology engineering purposes* because ORM has several strengths over other methods [H01]: ORM is fairly comprehensive in its treatment of many “practical” and “standard” rules, (e.g. identity, mandatory, uniqueness, subtyping, subset, equality, exclusion, frequency, transitive, acyclic, etc.). Furthermore, ORM has an expressive and stable graphical notation since it captures many rules graphically and it minimizes the impact of change on the models[78]. ORM has well-defined formal semantics (see e.g. [H89] [BHW91] [HPW93] [T96] [TM95] [HP95]). In addition, it is perhaps worthwhile to note that ORM derives from NIAM (Natural Language Information Analysis Method), which was explicitly designed to play the role of a stepwise methodology, to arrive at the "semantics" of a business application's data based on natural language communication.

5.2 ORM-Markup Language

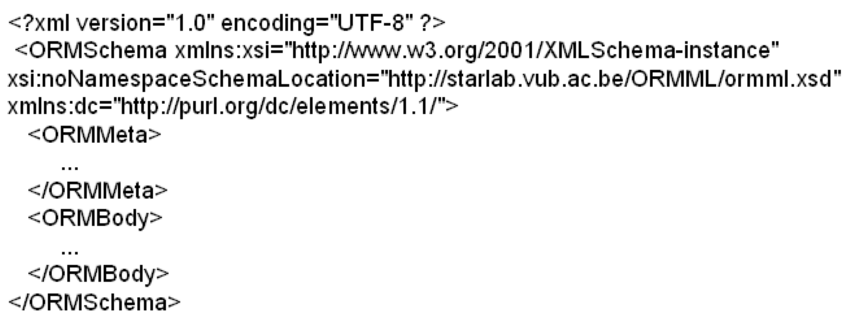
This section presents the ORM markup language (ORM-ML). ORM-ML is based on the XML syntax, and is defined in an XML-Schema (provided in Appendix A) that acts as its complete and formal grammar. Hence, any ORM-ML document should be valid according to this XML-Schema.

ORM-ML is not meant to be written by hand or interpreted by people. It is meant to be implemented for example, as a “save as” or “export to” functionality in ORM tools. This shall be illustrated in the next chapter as a functionality of our tool prototype.

In what follows, we describe the main elements of the ORM-ML grammar and demonstrate it using a few elementary examples. A more complete example is provided in Appendix A3. We chose to respect the ORM structure as much as possible by not “collapsing” it through the usual relational transformer that comes with most ORM-based tools. ORM-ML allows the representation of any ORM schema without a loss of information or a change in semantics, except for the geometry and topology (graphical layout) of the schema (e.g. location and shapes of the symbols) We include this in a separate graphical style sheet from that of the ORM Schema (see Appendix B2).

[78] In comparison with other approaches (e.g. ER, UML), ORM models are attribute-free; so they are immune from changes that cause attributes to be remodeled as entity types or relationships.

We represent the ORM document as a one node element called the ORMSchema, which consists itself of two nodes: ORMMeta and ORMBody. Fig. 5.1 shows an “empty” instance of this schema.



```
<?xml version="1.0" encoding="UTF-8" ?>
 <ORMSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://starlab.vub.ac.be/ORMML/ormml.xsd"
xmlns:dc="http://purl.org/dc/elements/1.1/">
  <ORMMeta>
     ...
  </ORMMeta>
  <ORMBody>
     ...
  </ORMBody>
</ORMSchema>
```

Fig. 5.1. An empty instance of the ORMSchema, as an example of ORM-ML document.

5.2.1 ORM-ML metadata

As a header to an ORM-ML document, an ORMMeta node includes metadata elements about the ORM document, such as ‘Title’, ‘URI’, ‘Creator’, ‘Version’, etc. A ORMMeta node consists of a set of Meta elements. Each Meta element has two attributes: name and content. The main idea of this *elementary* structure is to enable the flexibility of adopting existing metadata standards. For example, one may use the 15 well-known Dublin Core Meta elements[79] - an example of their use appears in fig. 5.12.

[79] The Dublin Core Metadata Initiative (http://www.dublincore.org , June 2004) is a cross-disciplinary international effort to develop mechanisms for the discovery-oriented description of diverse resources in an electronic environment. The Dublin Core Element Set comprises 15-elements which together capture a representation of essential aspects related to the description of resources. These 15-elements are namely: title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage and rights.

```
….
<ORMMeta>
  <Meta name="DC.Title" content="Complaint Problems"/>
  <Meta name="DC.Creator"  content="Mustafa Jarrar"/>
  <Meta name="DC.Language"  content="English"/>
  <Meta name="DC.Description" content="An axiomatization of
complaint problems categories…"/>
  …
</ORMMeta>
 ….
```

Fig. 5.2. An example of an ORMMeta node, using Dublin Core metadata elements.

To enable the foundation of libraries of application axiomatizations, we have developed a decent set of 25 metadata elements that better suit the description of ontological content. These elements are a specialization and extension of the Dublin Core elements. An example of this metadata appears in fig. 5.14. Appendix B1 presents a definition of these metadata elements[80]. We shall come back to this issue in the section 6.5 where we discuss the enabling of the development of "axiomatization libraries".

[80] It is perhaps worthwhile to note that our metadata elements (and their definitions) are adopted in the KnowledgeWeb Network of excellence project (KWEB EU-IST-2004-507482), and will be proposed as a standard for Ontology Metadata (or also called Ontology Registries). For more details, see [SGG+05].

-D

```
....
<ORMMeta>
  <Meta name="DM.Title" content="Complaint Problems"/>
  <Meta name="DM.Version"  content="v1.0"/>
  <Meta name="DM.Creator"  content="Mustafa Jarrar"/>
  <Meta name="DM.Genericity"  content="Application"/>
  <Meta name="DM.Language"  content="English"/>
  <Meta name="DM.DevelopmentStatus"  content="Final"/>
  <Meta name="DM.DomainSubject"  content="e-business, customer complaint"/>
  <Meta name="DM.CreationDate"  content="5/3/2003"/>
  <Meta name="DM.pecificationLanguage"  content="ORM-ML V1.2"/>
  <Meta name="DM.Tool"  content="DogmaModeler"/>
  <Meta name="DM.Application"  content="CCform"/>
  <Meta name="DM.Description" content="An axiomatization of complaint
problems categories…"/>
  …
</ORMMeta>
 ....
```

Fig. 5.3. An example of an ORMMeta Node, using DogmaModeler metadata elements.

5.2.2 ORM-ML Body

The ORMBody node consists of these five different (meta-ORM) elements: Object, Subtype, Predicate, Predicate_Object and Constraint.

Object Types

Object elements are abstract XML elements that are used to represent Object Types. They are identified by an attribute 'Name', which is the name of the Object Type in the ORM Schema, see fig. 5.4. Objects are implemented by two XML elements: LOT (Lexical Object Type, called Value Types in [H01]) and NOLOT (Non-Lexical Object Type, called Entity Types in [H01])[81]. LOT elements may have a *numeric* attribute, which is a boolean and indicates whether we deal with a numeric Lexical Object Type. NOLOT elements have a boolean attribute called *independent*, which indicates whether the Non Lexical Object Type is independent. NOLOT elements may also have a *reference* element. A

[81] *Informally speaking*, the idea of LOT and NOLOT in ORM, is similar the idea of ValueProperty and ObjectProperty in OWL. LOT represents ValueProperty, and NOLOT represents ObjectProperty.

-D

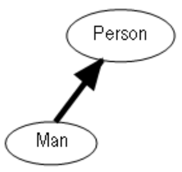
reference element would indicate how this NOLOT is identified by LOTs and other NOLOTs in a given application environment. A reference element has two attributes: ref_name (the name of the reference and numeric) and a boolean (to indicate whether it is a numeric reference).

```
...
<Object xsi:type="NOLOT" Name="Person"/>
...
```

Fig. 5.4. ORM-ML representation of an Object Type.

Subtypes

Subtype elements are used to represent subtype relationships between (non-lexical) object types. A subset element is required to have two elements: parent and child, where both refer to predefined object type elements. See fig. 5.5.



```
 ...
 <Object xsi:type="NOLOT" Name="Person"/>
 <Object xsi:type="NOLOT" Name="Man"/>
<Subtype>
  <Parent Object="Person" Role="Types"/>
  <Child Object="Man" Role="SubtypeOf"/>
</Subtype>
 ...
```

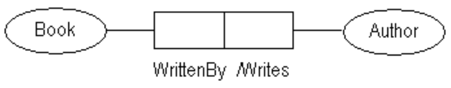
Fig. 5.5. ORM-ML representation of subtypes.

Predicates

Predicates consist of at least one Object_Role element. Such an element contains a reference to an object and may contain a role. They actually represent the rectangles in an ORM schema. Every Object_Role element needs a generated attribute 'ID' which identifies the Object_Role (see fig. 5.6). By using this ID attribute, we can refer to a particular Object_Role element in the rest of the XML document, which for example, we will need to do when we define constraints.

Predicates can have one or more rule elements. These elements can contain extra rules that are defined for the predicate.

Predicates also have two boolean attributes that are optional: '*Derived*' and '*Derived_Stored*' which indicate whether a predicate respectively is derived, or derived and stored, or not.

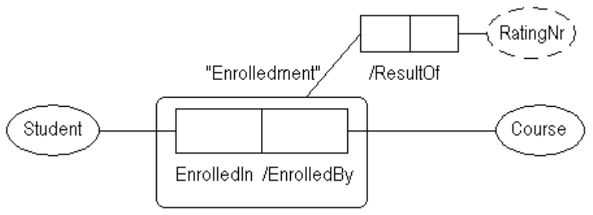


```
…
<Object xsi:type="NOLOT" Name="Book"/>
<Object xsi:type="NOLOT" Name="Author"/>
<Predicate>
   <Object_Role ID="ID1" Object="Book"
        Role="WrittenBy"/>
   <Object_Role ID="ID2" Object="Author"
        Role="Writes"/>
</Predicate>
…
```

Fig. 5.6. A simple binary predicate and its representation in ORM-ML.

Predicate Objects

Predicate_Objects are actually objectified predicates, which are used in nested fact types. They contain a predicate element and have an attribute called 'Predicate_Name'. So in fact, they are merely predicates that have received new object type names. In building Object_Roles, the Predicate_Name can be referenced. In this way we build predicates that contain objectified predicates instead of object types. See fig. 5.7.



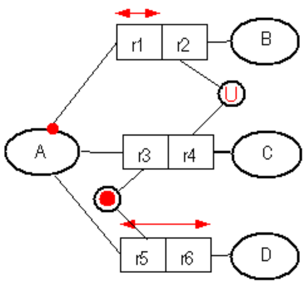
```
…
<Object xsi:type="NOLOT" Name=“Student"/>
<Object xsi:type=“NOLOT" Name=“Course"/>
<Object xsi:type="LOT" Name="RatingNr"/>
<Predicate_Object Predicate_Name=”Enrollment”>
  <Predicate>
    <Object_Role ID="ID18" Object=”Student” Role=”EnrolledIn”/>
    <Object_Role ID="ID19" Object=”Course” Role=”EnrolledBy”/>
  </Predicate>
</Predicate_Object>
<Predicate>
  <Object_Role ID="ID20" Object=“Enrollment" Role=""/>
  <Object_Role ID="ID21" Object="RatingNr“ Role=“ResultOf"/>
</Predicate>
…
```

Fig. 5.7. ORM-ML representation of nested fact types (Objectified predicates).

Constraints

Constraint elements represent the ORM constraints. The Constraint element itself is abstract, but it is implemented by different types of constraints, *viz.* Mandatory, Uniqueness, Subset, Equality, Exclusion, Value, Frequency, and Ring constraints. As mentioned above, we use the IDs of the Object_Role elements to define constraints.

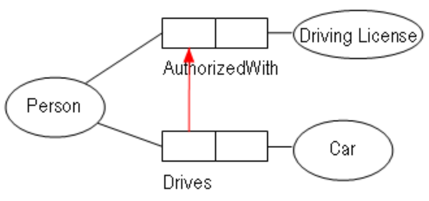
Uniqueness and mandatory constraint elements possess only Object_Role elements. These elements are the object_roles in the ORM diagram on which the constraint is placed. In this way, there is no need to make a distinction between the ORM-ML syntax of "external" and "internal" uniqueness constraints (see [H01]), or between mandatory and disjunctive mandatory constraints, see fig. 5.8.



```
…
<Predicate>  <Object_Role ID="ID1" Object="A" Role="r1"/>
  <Object_Role ID="ID2" Object="B" Role="r2"/> </Predicate>
<Predicate> <Object_Role ID="ID1" Object="A" Role="r3"/>
  <Object_Role ID="ID2" Object="C" Role="r4"/> </Predicate>
<Predicate>  <Object_Role ID="ID1" Object="A" Role="r5"/>
  <Object_Role ID="ID2" Object="D" Role="r6"/> </Predicate>
<Constraint xsi:type="Uniqueness">
<Object_Role>ID1</Object_Role> </Constraint>
<Constraint xsi:type="Uniqueness">
 <Object_Role>ID2</Object_Role>
<Object_Role>ID4</Object_Role> </Constraint>
<Constraint xsi:type="Uniqueness">
 <Object_Role>ID5</Object_Role>
 <Object_Role>ID6</Object_Role> </Constraint>
<Constraint xsi:type="Mandatory">
<Object_Role>ID1</Object_Role> </Constraint>
<Constraint xsi:type="Mandatory">
 <Object_Role>ID3</Object_Role>
<Object_Role>ID5</Object_Role> </Constraint>
…
```

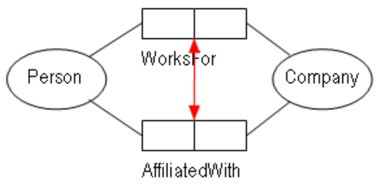
Fig. 5.8. ORM-ML representation of Uniqueness and Mandatory constraints.

The representation for *subset*, *equality*, and *exclusion* constraints is analogous, so we will only discuss them in general terms. Each of these constraints has references to (combinations of) object_role elements. For instance, to represent a subset constraint between two roles, we create a Subset element, containing two elements, Parent and Child. In the Parent element, we put references to the subsumed object_role, and in the Child element, we put references to the subsuming object_role. For equality and exclusion, we use First and Second elements instead of Parent and Child elements. Fig. 5.9., fig. 5.10, and fig. 5.11 show the ORM-ML representation of subset, equality, and exclusion constraints respectively.



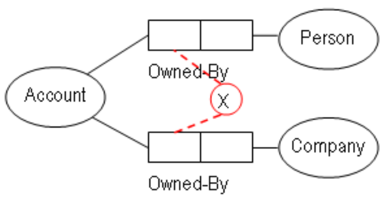
```
…
<Constraint xsi:type="Subset">
  <Parent>
    <Object_Role>ID1</Object_Role>
</Parent>
  <Child>
    <Object_Role>ID3</Object_Role>
  </Child>
</Constraint>
 ..
```

Fig. 5.9. ORM-ML representation of the Subset constraint.



```
…
<Constraint xsi:type="Equality ">
  <First>
    <Object_Role>ID1</Object_Role>
    <Object_Role>ID2</Object_Role>
  </First>
  <Second>
    <Object_Role>ID3</Object_Role>
    <Object_Role>ID4</Object_Role>
  </Second>
</Constraint>
…
```

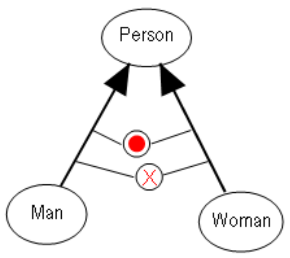
Fig. 5.10. ORM-ML representation of the Equality constraint.



```
…
<Constraint xsi:type="Exclusion ">
  <First>
    <Object_Role>ID1</Object_Role>
  </First>
  <Second>
    <Object_Role>ID3</Object_Role>
  </Second>
</Constraint>
…
```

Fig. 5.11. ORM-ML representation of the Exclusion constraint.

The representation for *Exclusive* and *Totality* constraints is analogous, and very simple. Each constrain has one supertype elements and (at least two) subtypes elements. See fig. 5.12.

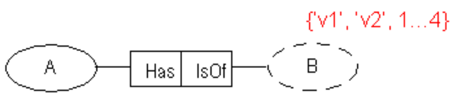


```
…
<Constraint xsi:type='Total'>
  <Supertype>Person</Supertype>
  <Subtype>Man</Subtype>
  <Subtype>Woman</Subtype>
</Constraint>
<Constraint xsi:type='Exclusive'>
 <Supertype>Person</Supertype>
  <Subtype>Man</Subtype>
  <Subtype>Woman</Subtype>
</Constraint>
…
```

Fig. 5.12. ORM-ML representation of the Exclusive and Totality constraint.

-D

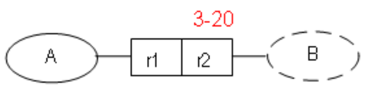
The *Value* constraint is represented in ORM-ML using the Value and ValueRange elements. The ValueRange element has two attributes: *begin* and *end*, with obvious meanings. Each of the Value and ValueRange elements have an additional attribute called "datatype" to indicate the datatype of the value. See fig. 5.13.



```
...
<Constraint xsi:type="Value" Object="B">
  <Value datatype="String">v1</Value>
  <Value Datatype="String">v2</Value>
  <ValueRange datatype="Number" begin="1" end="4"/>
</Constraint>
...
```

Fig. 5.13. ORM-ML representation of the value constraint.

The *Frequency* constraint is represented in ORM-ML by two attributes: Minimum and Maximum, which can defined on Object_Roles. See fig. 5.14.

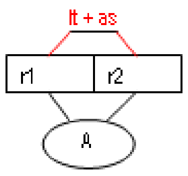


```
...
<Constraint xsi:type='Frequency' Minimum='3' Maximum='20'>
  <Object_Role>ID2</Object_Role>
</Constraint>
...
```

Fig. 5.14. ORM-ML representation of the Frequency constraint.

Finally, ring constraint elements are: antisymmetric (ans), asymmetric (as), acyclic (ac), irreflexive (ir), intransitive (it), symmetric (sym), acyclic+intransitive (ac+it), asymmetric+intransitive (as+it), intransitive+symmetric (it+sym), and irreflexive+symmetric (ir+sym). Ring constraint elements contain references to the object_roles they are put on. See Fig 5.15.

-D



```
...
<Constraint xsi:type='intransitive + Asymmetric'>
 <Object_Role>r1 </Object_Role>
 <Object_Role>r2</Object_Role>
</Constraint>
....
```

Fig. 5.15. ORM-ML representation of the Ring constraints.

Remark: ORM-ML also supports modular ORM schemes, which allows the representation of sub ORM schemes (seen as composed modules). We postpone the discussion of this issue to section 6.6.

5.3 Discussion and conclusions

In this chapter, we have presented the ORM markup language that represents ORM conceptual diagrams in an XML-based syntax. Our main goals of doing this are:

- *Enable the ORM conceptual diagrams to be shared, exchanged, and processed at run-time*. ORM-ML as a standardized syntax for ORM models may assist interoperation tools to exchange, parse or understand the ORM schemas. Like ORM-ML, any conceptual modeling approach (e.g. EER, UML, etc.) could have a markup language.

- *Enable conceptual data modeling methods to be (re)used for ontology engineering purposes.* Indeed, as we have discussed in section 3.4, conceptual data modeling methods suit many (or maybe most) application scenarios and usability perspectives. In addition, the large set of existing conceptual modeling methods, graphical notations, and tools can make ontologies better understandable, and easier to adopt, construct, visualize and verbalize. Legacy conceptual schemes can be mined and/or "ontologized". In the next chapter, we illustrate these issues by using ORM for modeling and representing application

-D

axiomatizations, which shall be defined in terms of domain axiomatizations (ontology base).

In addition, by standardizing such a markup language, several *other* advantage are worth noting:

- Interoperability for exchanging and sharing conceptual data models over the Internet. Facilities are needed to share and exchange ORM conceptual models in terms of a networked, distributed computing-driven, and collaborative environment, and to allow users to browse and edit shared knowledge over the Internet, intranets and other channels. A conceptual schema markup language provides a standardizable method to achieve interoperability among CASE tools that use the conceptual modeling technique.

- Implementing a conceptual query language over the Web. In open and distributed environments, the building of queries should be possible regardless of the internal representation of the data. Query languages based on ontologies (seen as shared conceptual models) help users not only to build queries, but also make them easier, more expressive, and more understandable than corresponding queries in a language like SQL. Exchanging, reusing, or sharing such queries efficiently between agents over the web is substantially facilitated by a standardized markup language. Consequently, ORM-based query languages (e.g. RIDL [VB82] [M81], ConQuer [BH96]) would gain from ORM-ML by representing queries in such an exchangeable representation.

- Building transformation style sheets. Building transformation style sheets for a given usage or need, for example, for the first order rewriting of formalisms of ORM-ML documents, or to transform the XML-based representation into another XML-based representation. Another important and strategic issue is that one could write a style sheet to generate the given ORM model instance into a given rule-engine's syntax, to allow for run-time interpretation by that rule engine. It could for instance, perform instance validation and integrity checks.

- Generating Verbalizations. The verbalization of a conceptual model is the process of writing its facts and constraints in pseudo natural language sentences. This assumedly allows non-experts to check, validate, or even build conceptual schemas. In the next chapter, we show how to generate the verbalization of ORM models by building a verbalization template (built as separate XML-based style sheets) parameterized over ORM-ML documents.

Having concluded this section, we proceed to present the DogmaModeler ontology engineering tool that constitutes the implementation section of this thesis.

Chapter 6

DogmaModeler Ontology Engineering Tool

"The new tools of ontological engineering might help us to realize Peirce's vision of a time when operations upon diagrams will take the place of the experiments upon real things that one performs in chemical and physical research."

-(Barry Smith, [S02])

In this chapter we present a prototype of an ontology engineering tool. In section 6.1, we give a quick overview of the tool. The illustration of how to model a domain and application axiomatizations will be presented in section 6.2 and section 6.3 respectively. In section 6.4, we give an overview of the validation types that are supported in the DogmaModeler. The DogmaModeler's support of axiomatization libraries is presented and discussed in section 6.5. In section 6.6., we present the implementation of module composition. The other functionalities of DogmaModeler will be briefly explained in section 6.7. To end, some conclusions and final remarks are made in section 6.8.

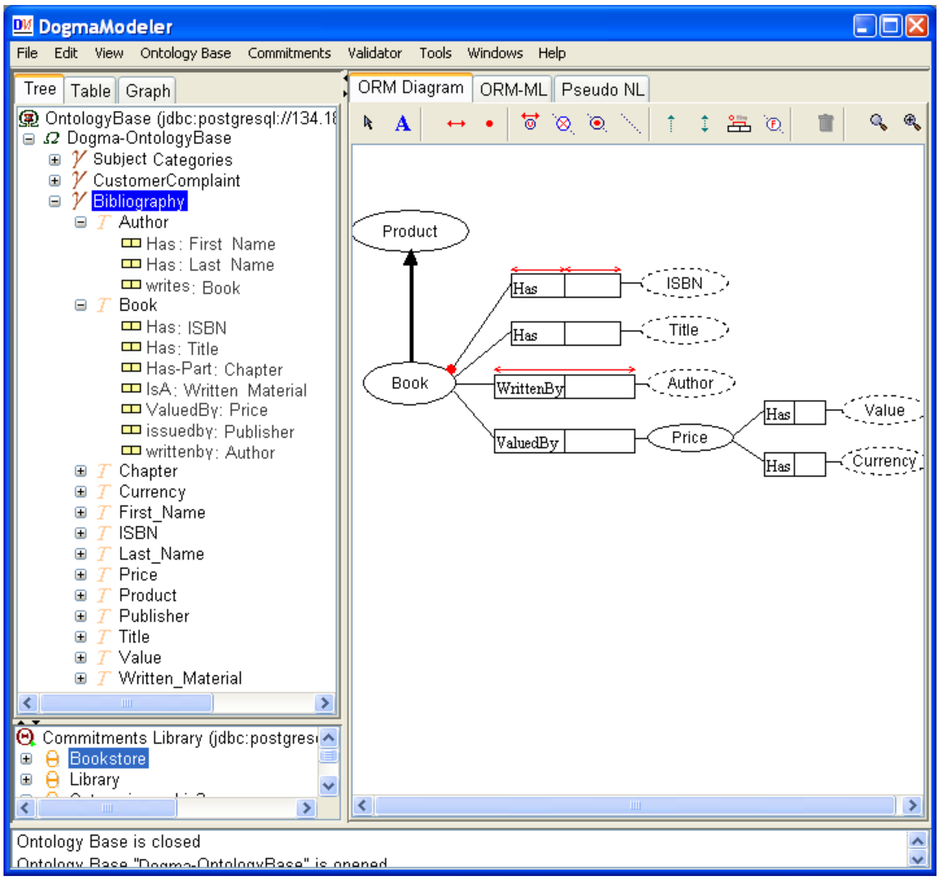
6.1 Introduction, a quick overview of DogmaModeler

This section briefly outlines our DogmaModeler tool prototype for ontology engineering. Its implementation is based on the methodological principles described in this thesis.

The DogmaModeler supports the following functionalities (among other things that shall be illustrated later):

- Modeling, browsing, and managing both domain and application axiomatizations;
- Modeling application axiomatizations using the ORM graphical notation, and generating the corresponding ORM-ML automatically;
- Verbalizing application axiomatizations into pseudo natural language (supporting flexible verbalization templates, for e.g. English, Dutch, Arabic, and Russian);
- Automatic composition of axiomatization modules;
- Validations of the syntax and semantics of axiomatizations;
- An illustration is given of the process of incorporating lexical resources in ontology modeling; in order to the support the modeling process of glosses;
- A simple approach to support the multilingual lexicalization of ontologies;
- Automatic mapping of ORM schemes into X-Forms and HTML-Forms.

Fig. 6.1 shows a screenshot of DogmaModeler. Notice its three main windows: the ontology base window, the commitment modeling window, and the commitment library window.



Ontology Base is closed
Ontology Base "Dogma-OntologyBase" is opened

Fig. 6.1. A general screenshot of DogmaModeler.

Ontology base window (the top left side of fig. 6.1)

Before building ontological commitments (i.e. application axiomatization), ontology builders should define their lexons in the ontology base window, in case it is empty. This window presents the set of lexons -{< γ: Term$_1$, Role, InvRole, Term$_2$>}- in a tree-like structure[82]. The first level, (***Ω***) represents ontology bases (e.g. Dogma-Ontologybase). In the second level, each node (γ) represents a context (e.g. Bibliography).

[82] The ontology base tree has advanced features, so it can also be browsed and seen as a graph.

Notice that level 0 (🖥) in the tree represents the ontology base server, where the content of the ontology bases is hosted and managed. All transactions carried out at the ontology base (e.g. creating contexts, editing lexons) will be transmitted, verified and executed on the server.

Notice that level 0 (🖥) in the tree represents the ontology base server, where the content of ontology bases is hosted and managed. All transactions on the ontology base (e.g. creating contexts, editing lexons) will be transmitted, verified and executed on the server.

Commitment modeling window (the right side of fig. 6.1)

This window consists of three panels: ORM, ORM-ML, and Pseudo NL. To build an application axiomatization, ontology builders can drag and drop lexons from the ontology base window into the ORM panel (to define the ontological view). When doing so, lexons will be mapped automatically into ORM fact types. Then, in order to define constraints on these lexons, ontology builders can use the ORM family of constraints (see icons in the top of the ORM panel).

Commitment library window (Under the ontology base window)

The purpose of this window is to enhance the reusability, management, and organization of application axiomatizations. The current implementation allows ontology builders to access and browse application axiomatizations stored in a library (Θ). Each node (⊖) in the first level of the tree represents an application axiomatization. By expanding an axiomatization node, the set of lexons and the set of constraints that are subject to this axiomatization will appear in the second level.

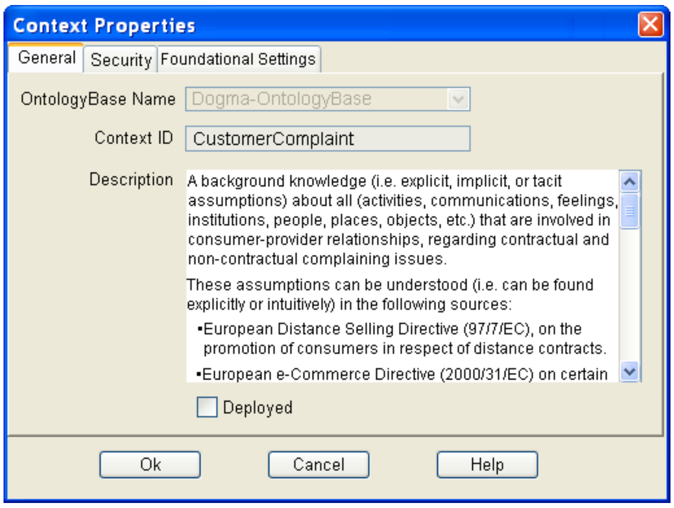
Remark: Although in this chapter, we sometimes describe "how" to model an ontology using the DogmaModeler, our description is intended neither to be a stepwise methodology nor to serve as a manual of the DogmaModeler.

6.2 Modeling domain axiomatizations in the Ontology Base

In this section we present how domain axiomatizations can be developed and represented in the ontology base. We present how, for modeling purposes, the DogmaModeler supports: Context, Lexon, Term, Gloss, and Role/InvRole. These are the main building blocks of a domain axiomatization.

6.2.1 Context Modeling

The first step to developing a domain axiomatization is to specify the context(s) of the domain. In other words, providing information about the scope of the axiomatization, in which the interpretation (i.e. the intended meaning) of the ontology terminology is bounded. In the DogmaModeler, each context should have a Context ID, and a Context Description. Fig. 6.2 shows the context modeling window and an example of modeling the 'CustomerComplaint' context of the CContology[83].



OntologyBase Name: Dogma-OntologyBase

Context ID: CustomerComplaint

Description: A background knowledge (i.e. explicit, implicit, or tacit assumptions) about all (activities, communications, feelings, institutions, people, places, objects, etc.) that are involved in consumer-provider relationships, regarding contractual and non-contractual complaining issues.

These assumptions can be understood (i.e. can be found explicitly or intuitively) in the following sources:

- European Distance Selling Directive (97/7/EC), on the promotion of consumers in respect of distance contracts.
- European e-Commerce Directive (2000/31/EC) on certain

Fig. 6.2. Context modeling window.

[83] This ontology, and its 'CustomerComplaint' context, shall be present in more detail in chapter 7.

In the Context Description field, one may refer to sources such as a set of documents, laws, regulations and informal descriptions of "best practices". The idea is that the interpretation of the terms that will appear in the lexons within this context is bounded to concepts that might be referred to (explicitly or *intuitively*) within these resources. Lexons are assumed to be "true within their context's source".

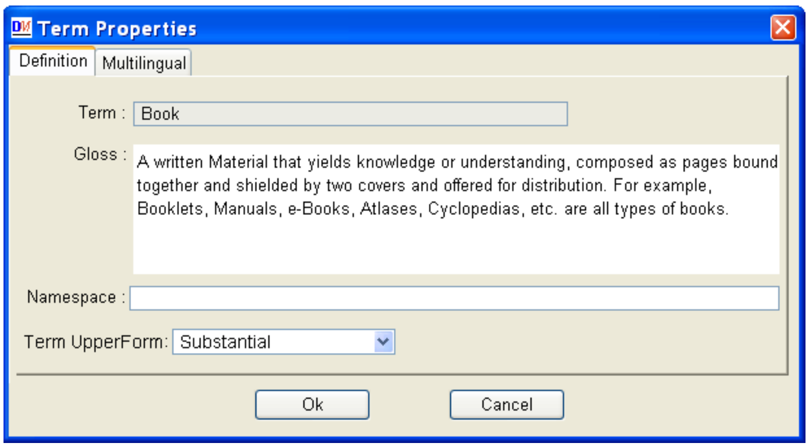
If an ontology is mined from a corpse of documents, the recommended best practice is to cite these documents in the context description. In case an ontology is developed based on (or conforming to) a set of laws, regulations or constitutions, these rules should be cited.

From a methodological viewpoint, by describing their context, ontology builders will be encouraged to decide the scope and coverage of their axioms, especially in the early development phases. A context description (and the resources cited in it) can be also used for investigating the correctness of glosses and lexons.

The "Deployed" flag, at the bottom of the context modeling window indicates whether the lexons in this context are "being used" or are still "under development". If a context is flaged as deployed, the DogmaModeler disables all delete and change functions over the properties of all terms, roles, and lexons.

6.2.2 Concept Modeling

When introducing a new concept (i.e. a term within a given context), ontology builders should define its gloss. Fig. 6.3 shows the concept-modeling window with an example of the term 'Book' and its gloss, within the context of a 'Bibliography'. See our methodological guidelines for gloss-modeling in section 3.3.6.


Term Properties

Definition | Multilingual

Term : Book

Gloss : A written Material that yields knowledge or understanding, composed as pages bound together and shielded by two covers and offered for distribution. For example, Booklets, Manuals, e-Books, Atlases, Cyclopedias, etc. are all types of books.

Namespace :

Term UpperForm: Substantial

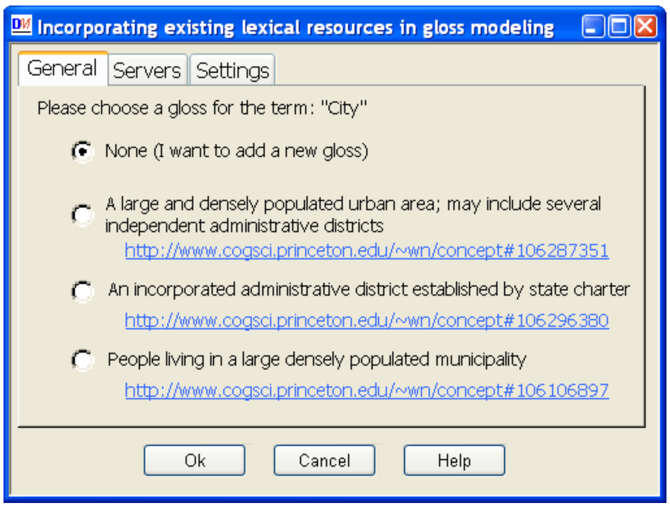
Fig. 6.3. Concept modeling window.

Incorporating existing lexical resources in gloss modeling

As we have discussed in section 3.5, many existing lexical resources (such as lexicons, glossaries, thesaurus, dictionaries, etc.) are indeed important sources of glosses. To enable the adoption and reusability of such resources, fig. 6.4 shows a screenshot of a menu of glosses of the term 'City', which are retrieved from WordNet. The idea is that after introducing a new term, the DogmaModeler automatically offers a menu of glosses for this term. Ontology builders can then, choose or define a new gloss. If an existing gloss has been chosen, a reference to this gloss is recorded in the "Namespace" field[84].

[84] Because of time limitations, this functionality is not yet fully implemented in the DogmaModeler.

-D


Incorporating existing lexical resources in gloss modeling

General | Servers | Settings

Please choose a gloss for the term: "City"

(●) None (I want to add a new gloss)

( ) A large and densely populated urban area; may include several independent administrative districts
http://www.cogsci.princeton.edu/~wn/concept#106287351

( ) An incorporated administrative district established by state charter
http://www.cogsci.princeton.edu/~wn/concept#106296380

( ) People living in a large densely populated municipality
http://www.cogsci.princeton.edu/~wn/concept#106106897

Fig. 6.4. Incorporating existing lexical resources in gloss modeling.

Recall that the notion of gloss is not intended to catalog general information or to provide morphological issues about a term, as conventional dictionaries usually do. As we have discussed in section 3.3.6, a gloss has a strict intention in our appraoch and not just any lexical resource can be adopted. The lexicon should provide a clear discrimination of word/term meaning(s) in a machine-referable manner, much like the synsets in WordNet.

The "Upper Form" field in the concept-modeling window serves to declare the term-upper-form of the concept. For example, the Upper Form of 'Book' is 'Substantial' according to the DOLCE foundational ontology. See our earlier discussion on this issue in section 3.3.7. The full incorporation of upper level (foundational) ontologies in the DogmaModeler is considered a future development task.

6.2.3 Lexon Modeling

Lexons are the main axioms in a domain axiomatization. Recall that a lexon has the form: <Context: Term$_1$, Role, InvRole, Term$_2$> (see section 3.3.). After having introduced a term and its informal definition (i.e. gloss) into the ontology base, ontology builders can introduce lexons. Fig. 6.5 shows a simplified lexon-modeling window[85]. In this window, for a Term, within a Context, ontology builders may declare a lexon by introducing its Role, InvRole, Term$_2$, and then choose the LexonUpperForm of this lexon.

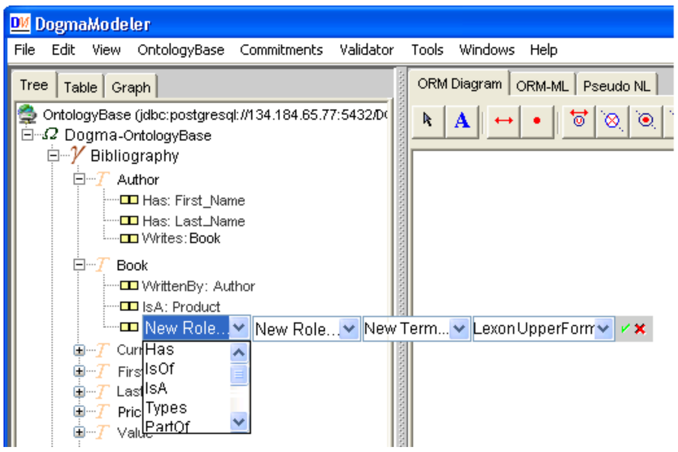


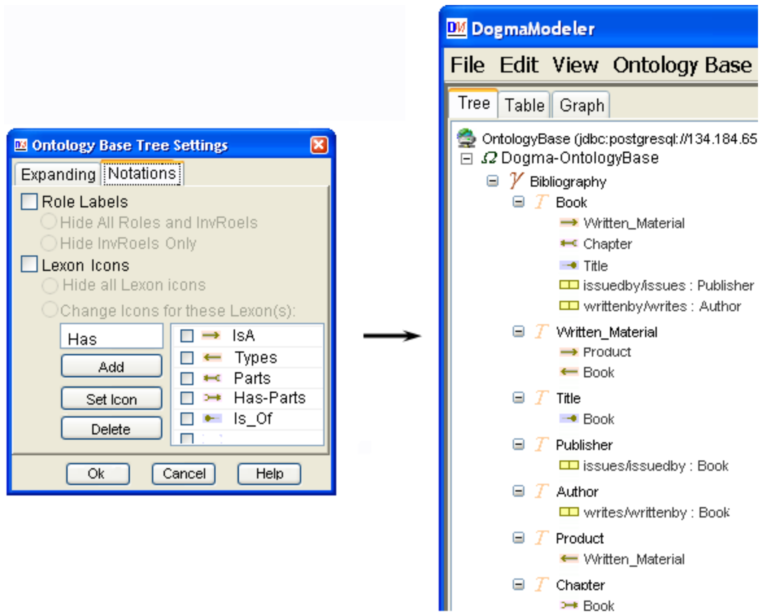
Fig. 6.5. Lexon-modeling window.

The "lexonUpperFrom" field allows ontology builders to declare the primitive lexon type (Subsumption, Parthood, Dependence, Property-of, Attribution etc.), thus committing to an upper level ontology of relationship kinds, also known as "basic primitive relationships" (see our discussion on this issue in section 3.3.7.). As the incorporation of upper level ontologies in our approach is still in progress, the DogmaModeler's full support of the LexonUpperForm, is considered a future development task. At this stage, the DogmaModeler does not impose any restriction on this field.

Lexon notation and visualization

[85] DogmaModeler supports another more sophisticated window for modeling lexons, which allows faster and more scalable (search and retrieval) of existing terms and roles. However, this feature is not presented in this section for the sake of simplicity.

To simplify the lexon modeling process, the DogmaModeler allows users to customize the lexon graphical notation. As apparent in the left side of fig. 6.6, users may choose to hide Role/InvRole labels; and/or they may introduce their own graphical notation (i.e. lexon icons).



Role Labels
Hide All Roles and InvRoels
Hide InvRoels Only
Lexon Icons
Hide all Lexon icons
Change Icons for these Lexon(s):
Has
IsA
Types
Parts
Has-Parts
Is_Of

OntologyBase (jdbc:postgresql://134.184.65
Dogma-OntologyBase
Bibliography
Book
Written_Material
Chapter
Title
issuedby/issues : Publisher
writtenby/writes : Author
Written_Material
Product
Book
Title
Book
Publisher
issues/issuedby : Book
Author
writes/writtenby : Book
Product
Written_Material
Chapter
Book

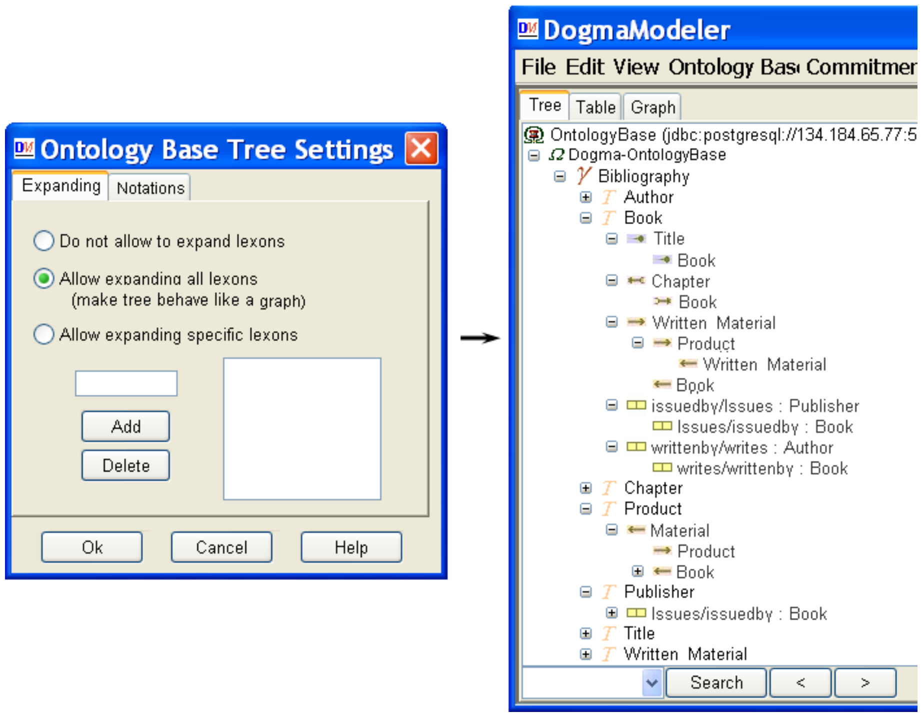
Fig. 6.6. Lexon graphical notation.

In the current version of the DogmaModeler, users have the freedom to upload and change any lexon icons according to their preference. However, we plan to restrict this facility by reserving an icon for each lexon kind. Each lexon notation will have fixed semantics and this will commit to an upper level ontology of relationship kinds.

To simplify the lexon browsing process, DogmaModeler allows users to customize the browsing settings of the lexon tree. As shown in the left side of fig. 6.7, users may choose to expand lexon nodes, so that one can browse the tree as one browses a graph. In the same way, users may also choose to expand *only* a specific kind(s) of lexons. For example, one may

-D

wish to expand only lexons that denote transitive-alike relationships such as subsumption or parthood.



Do not allow to expand lexons
[✓ Allow expanding all lexons (make tree behave like a graph)]
Allow expanding specific lexons

```
OntologyBase (jdbc:postgresql://134.184.65.77:5
  Dogma-OntologyBase
    Bibliography
      Author
      Book
        Title
          Book
        Chapter
          Book
        Written  Material
          Product
            Written  Material
          Book
        issuedby/Issues : Publisher
          Issues/issuedby : Book
        writtenby/writes : Author
          writes/writtenby : Book
      Chapter
      Product
        Material
          Product
          Book
      Publisher
        Issues/issuedby : Book
      Title
      Written  Material
```

Fig. 6.7. Lexon browsing.

By selecting “Allow expanding lexons…”, users will be able to expand T_2 of the Lexon. The expansion will show all lexons where this T_2 is T_1 for other lexons, and so on. In this way, users will be able to browse the tree as they browse a graph. Note that expanding a node that is already expanded in the same sub-tree (i.e. cycle) is not possible. If such an attempt is made, the focus of the window will be moved to the previous sub-tree. As a result, users will be able to visit all lexons starting from any Term (see the right side of fig. 6.7.).

Remark: although the tree-alike representations of lexons are very simple and easy for ontology builders to understand, the main disadvantage of such a representation is scalability. Browsing large-scale ontology bases

-D

in this way is obviously not convenient as it requires ontology builders to perform many search and expand operations and browsing tree-alike representations is scalable up to several hundred terms or lexons.

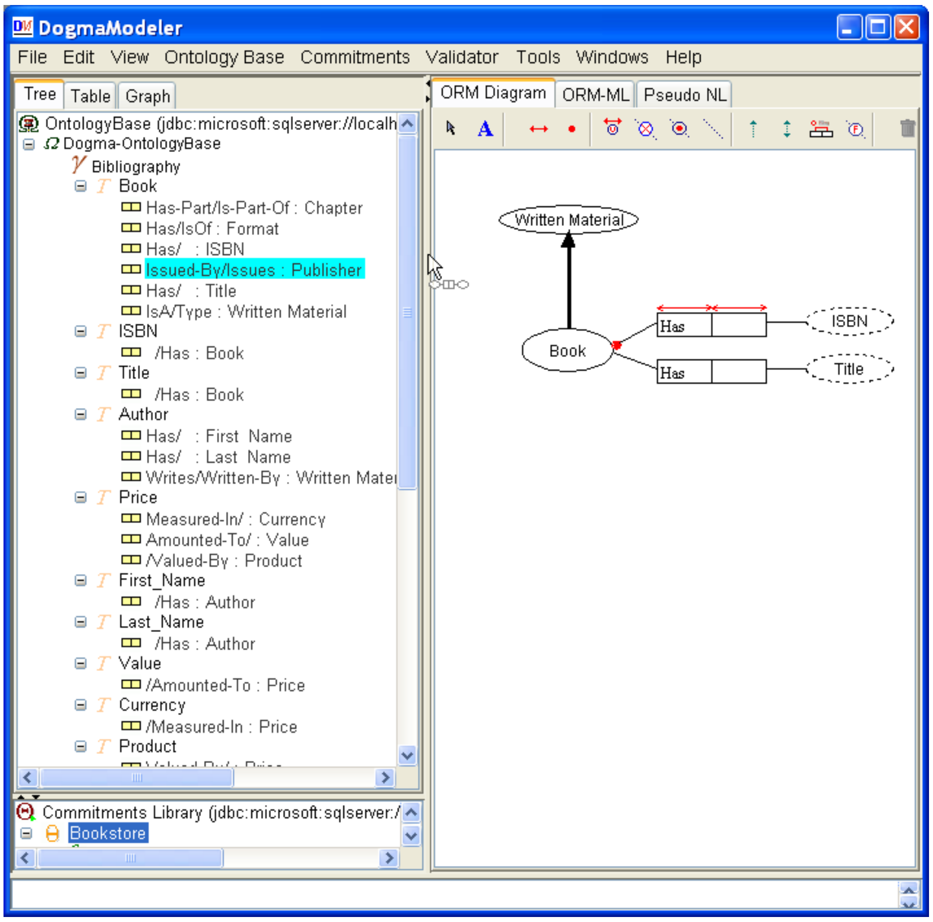
Nevertheless, several techniques can be used for modeling, browsing, and visualizing ontology bases, but these are major research topics on their own. A promising technique that we plan to incorporate in future is called LexoVis [P05] as this technique seems to allow scalable visualization of lexons.

6.3 Modeling application axiomatizations

While an ontology base is intended to be a shared and public axiomatization (characterizing its intended models) at the domain level, application axiomatizations are intended to be local and highly usable at the task/application-kind level. Given an ontology base, applications that are interested only in a subset of the intended models of a concept in accordance with their usability perspective are supposed to provide some rules to specialize these intended models. As we have discussed in chapter 3, we require that the vocabulary used in application axiomatizations be restricted to the vocabulary defined in its ontology base. An application axiomatization becomes a set of rules to constrain the particular use of the domain vocabulary.

As particular applications commit to the ontology base through application axiomatization(s), such axiomatizations are seen as (and also called) the application's ontological commitment (see section 3.4.).

The process of modeling such ontological commitments in the DogmaModeler is designed to be very simple. As appears in fig. 6.8, ontology builders can drag and drop lexons from the ontology base window into the ORM Diagram panel. These lexons are mapped and drawn automatically as fact-types, according to the ORM notation. Ontology builders then can define new constraints on these lexons (see the icons of the ORM family of constraints in the top of the ORM Diagram panel).


DogmaModeler
File Edit View Ontology Base Commitments Validator Tools Windows Help
Tree Table Graph
ORM Diagram ORM-ML Pseudo NL
Written Material
Book
Has
Has
ISBN
Title

OntologyBase (jdbc:microsoft:sqlserver://localh
Dogma-OntologyBase
Bibliography
Book
Has-Part/Is-Part-Of : Chapter
Has/IsOf : Format
Has/ : ISBN
Issued-By/Issues : Publisher
Has/ : Title
IsA/Type : Written Material
ISBN
/Has : Book
Title
/Has : Book
Author
Has/ : First Name
Has/ : Last Name
Writes/Written-By : Written Mate
Price
Measured-In/ : Currency
Amounted-To/ : Value
/Valued-By : Product
First_Name
/Has : Author
Last_Name
/Has : Author
Value
/Amounted-To : Price
Currency
/Measured-In : Price
Product

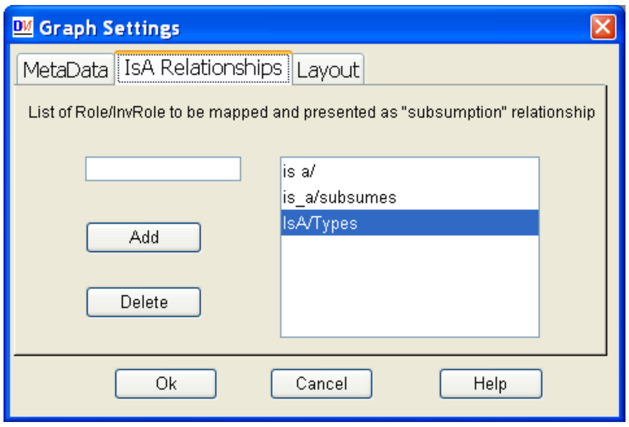
Commitments Library (jdbc:microsoft:sqlserver:/
Bookstore

Fig. 6.8. Modeling application axiomatizations.

The mapping of lexons as intuitive domain axioms into ORM fact-types that have predefined formal semantics [V82] is done as follows: a Term within a context is mapped directly into an Object-Type in ORM and Roles within a lexon are also mapped directly into ORM Roles within a fact-type. In the case of ORM Subtype relations that have specific "build-in" semantics, ontology builders need to customize the "Graph settings" window in order to specify which roles should be mapped (see fig. 6.9.).

-D

The DogmaModeler does not support ORM unary roles and nested fact types.



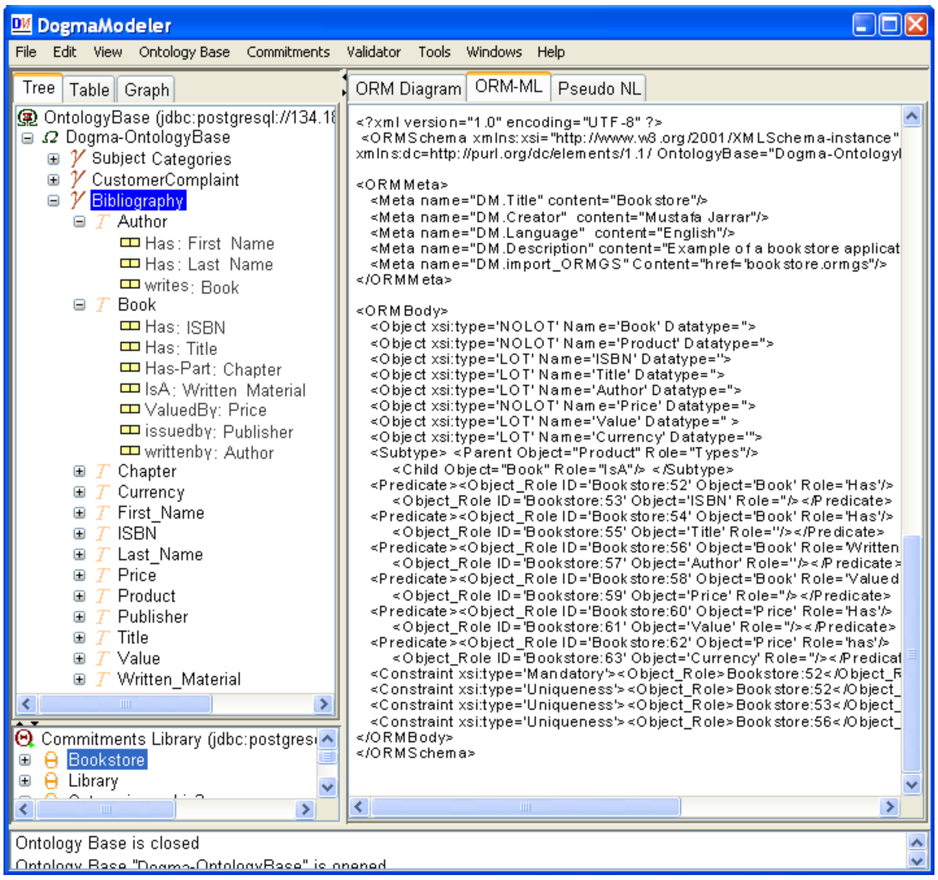
List of Role/InvRole to be mapped and presented as "subsumption" relationship

is a/
is_a/subsumes
IsA/Types

Fig. 6.9. Mapping to ORM Subtype relationship.

6.3.1 Generating ORM-ML

Fig. 6.10 shows the ORM markup language corresponding to the ORM diagram in Fig. 6.1. This language is automatically generated by the tool. The DogmaModeler supports import-export ORM-ML into text files, and downloads or uploads it into the ontology server.


DogmaModeler
File Edit View Ontology Base Commitments Validator Tools Windows Help
Tree Table Graph
ORM Diagram ORM-ML Pseudo NL

```
OntologyBase (jdbc:postgresql://134.18
  Dogma-OntologyBase
    Subject Categories
    CustomerComplaint
    Bibliography
      Author
        Has: First Name
        Has: Last Name
        writes: Book
      Book
        Has: ISBN
        Has: Title
        Has-Part: Chapter
        IsA: Written Material
        ValuedBy: Price
        issuedby: Publisher
        writtenby: Author
      Chapter
      Currency
      First_Name
      ISBN
      Last_Name
      Price
      Product
      Publisher
      Title
      Value
      Written_Material

Commitments Library (jdbc:postgres
  Bookstore
  Library
```

```
<?xml version="1.0" encoding="UTF-8" ?>
 <ORMSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dc=http://purl.org/dc/elements/1.1/ OntologyBase="Dogma-Ontology

<ORMMeta>
  <Meta name="DM.Title" content="Bookstore"/>
  <Meta name="DM.Creator"  content="Mustafa Jarrar"/>
  <Meta name="DM.Language"  content="English"/>
  <Meta name="DM.Description" content="Example of a bookstore applicat
  <Meta name="DM.import_ORMGS" Content="href='bookstore.ormgs"/>
</ORMMeta>

<ORMBody>
  <Object xsi:type='NOLOT' Name='Book' Datatype=">
  <Object xsi:type='NOLOT' Name='Product' Datatype=">
  <Object xsi:type='LOT' Name='ISBN' Datatype=">
  <Object xsi:type='LOT' Name='Title' Datatype=">
  <Object xsi:type='LOT' Name='Author' Datatype=">
  <Object xsi:type='NOLOT' Name='Price' Datatype=">
  <Object xsi:type='LOT' Name='Value' Datatype=" >
  <Object xsi:type='LOT' Name='Currency' Datatype='">
  <Subtype> <Parent Object="Product" Role="Types"/>
     <Child Object="Book" Role="IsA"/> </Subtype>
  <Predicate><Object_Role ID='Bookstore:52' Object='Book' Role='Has'/>
     <Object_Role ID='Bookstore:53' Object='ISBN' Role="/></Predicate>
  <Predicate><Object_Role ID='Bookstore:54' Object='Book' Role='Has'/>
     <Object_Role ID='Bookstore:55' Object='Title' Role=''/></Predicate>
  <Predicate><Object_Role ID='Bookstore:56' Object='Book' Role='Written
     <Object_Role ID='Bookstore:57' Object='Author' Role=''/></Predicate>
  <Predicate><Object_Role ID='Bookstore:58' Object='Book' Role='Valued
     <Object_Role ID='Bookstore:59' Object='Price' Role="/></Predicate>
  <Predicate><Object_Role ID='Bookstore:60' Object='Price' Role='Has'/>
     <Object_Role ID='Bookstore:61' Object='Value' Role="/></Predicate>
  <Predicate><Object_Role ID='Bookstore:62' Object='Price' Role='has'/>
     <Object_Role ID='Bookstore:63' Object='Currency' Role="/></Predicat
  <Constraint xsi:type='Mandatory'><Object_Role>Bookstore:52</Object_R
  <Constraint xsi:type='Uniqueness'><Object_Role>Bookstore:52</Object_
  <Constraint xsi:type='Uniqueness'><Object_Role>Bookstore:53</Object_
  <Constraint xsi:type='Uniqueness'><Object_Role>Bookstore:56</Object_
</ORMBody>
</ORMSchema>
```

```
Ontology Base is closed
Ontology Base "Dogma-OntologyBase" is opened
```

Fig. 6.10. The ORM-ML panel window.

The graphical layout of the ORM diagrams (shapes, positions, color, etc.) is generated by the DogmaModeler into a separate XML-based document, called the ORM graphical style-sheet. The XML-schema of these graphical style-sheets is presented in Appendix B2.

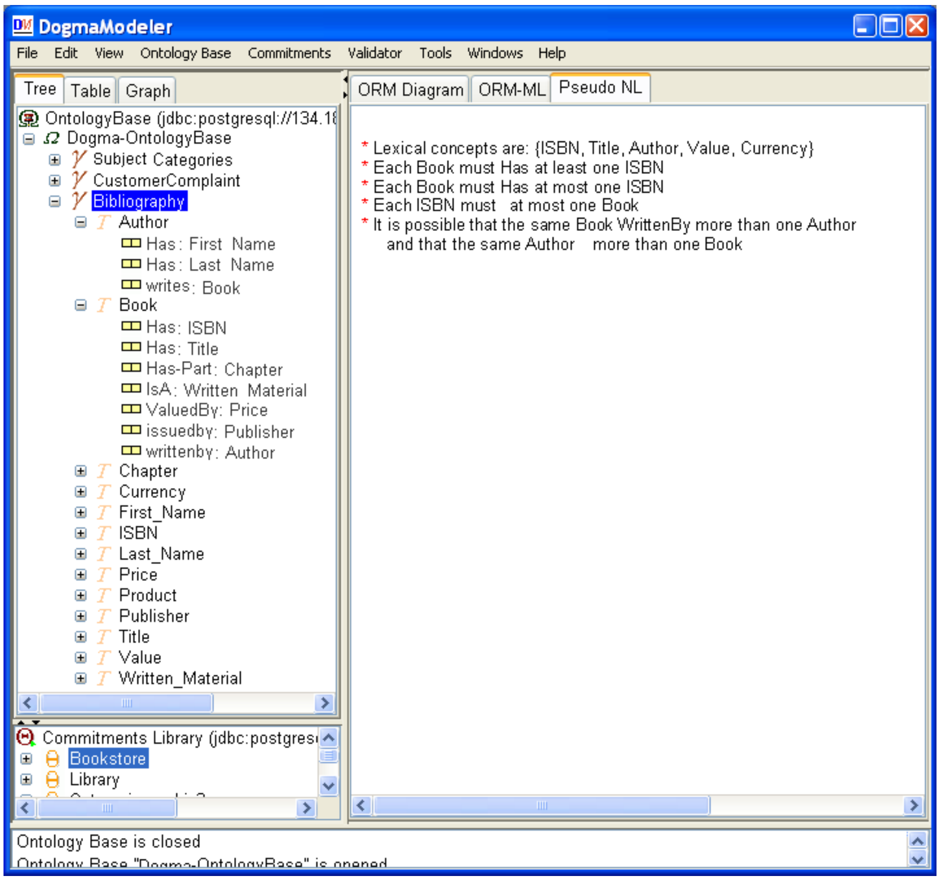
6.3.2 Verbalization[86]

Fig. 6.11 shows a verbalization of the ORM diagram presented in Fig. 6.1. This verbalization is a pseudo natural language (fixed-syntax English sentences) generated automatically by the DogmaModeler. The

[86] Later, this section was revised and extended, see [JKD06].

-D

DogmaModeler generates such a verbalization by applying predefined verbalization templates parameterized over an ORM-ML document.



* Lexical concepts are: {ISBN, Title, Author, Value, Currency}
* Each Book must Has at least one ISBN
* Each Book must Has at most one ISBN
* Each ISBN must at most one Book
* It is possible that the same Book WrittenBy more than one Author and that the same Author more than one Book

Ontology Base is closed
Ontology Base "Dogma-OntologyBase" is opened

Fig. 6.11. The Pseudo NL panel window.

In our experience[87], verbalizations greatly assists non-ontology-experts in building and validating axiomatizations. It is indeed an easily understood language for domain experts, especially those who are not trained to understand technical or formal languages. Although it is not a formal

[87] ie: Specifically, our experience in building the CContology in cooperation with many domain experts (about 40 layers, application expertise, etc.). We shall report this experience in greater detail in chapter 7.

-D

language, verbalization templates should be unambiguous and well structured.

The DogmaModeler supports flexible and multilingual verbalization techniques. We have developed an easy-to-customize verbalization template to verbalize ORM-ML documents. We have translated this template into several languages. If the content of an ORM-ML document is lexicalized in Italian for example, the DogmaModeler is able to generate the verbalizations in Italian. Appendix B3 presents five verbalization templates in English, Dutch, Arabic and Russian[88]. In the following paragraphs, we illustrate our English verbalization template using selected examples.

Fig. 6.12 shows the verbalization template of the Mandatory constraint. Given this template, the verbalization of the mandatory constraint in fig. 6.13 is: "Each *Book* must *Has* at least one *ISBN*".

```
<Constraint xsi:type="Mandatory">
   <Text> Each</Text>
   <Object index="0"/>
   <Text> must</Text>
   <Role index="0"/>
   <Text> at least one</Text>
   <Object index="1"/>
</Constraint>
```

Fig. 6.12. Verbalization template for the ORM Mandatory constraint.

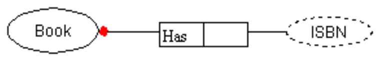
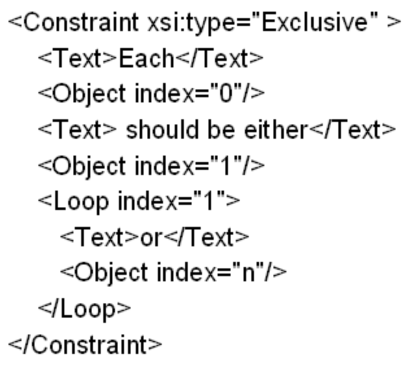


Fig. 6.13. Example of ORM mandatory constraint.

Keeping in mind the verbalization template of Exclusive constraint in fig. 6.14, the verbalization of the constraint in fig. 6.15 reads: "Each

[88] The support of more languages is designed to be very simple. It requires just the provision of a new temple for the language.

Complaint Resolution should be either *Economic Resolution* or *Symbolic Resolution* or *Information Correction*”.



```
<Constraint xsi:type="Exclusive" >
   <Text>Each</Text>
   <Object index="0"/>
   <Text> should be either</Text>
   <Object index="1"/>
   <Loop index="1">
     <Text>or</Text>
     <Object index="n"/>
   </Loop>
</Constraint>
```

Fig. 6.14. Verbalization template for the ORM Exclusive constraint.

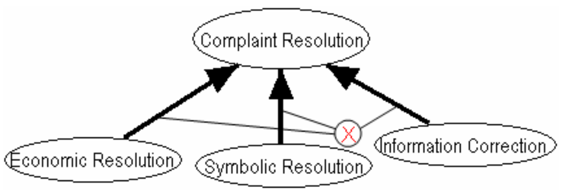


Fig. 6.15. Example of an ORM Exclusive constraint.

Given the verbalization template of the Subset constraint in fig. 6.16, the verbalization of the constraint in fig. 6.17 should read: “If a *Person Drives* a *Car* then this *Person* must be *AuthorizedWith* a *Driving License*”.

```
<Constraint xsi:type="Subset" >
    <Text>If a </Text>
    <Object  index="0" />
    <Role index="child" />
    <Text>a</Text>
    <Object index="child" />
    <Text>then this</Text>
    <Object index="0" />
    <Text>must be</Text>
    <Role index="parent" />
    <Text>a</Text>
    <Object index="parent" />
</Constraint>
```

Fig. 6.16. Verbalization template for the ORM Subset constraint.

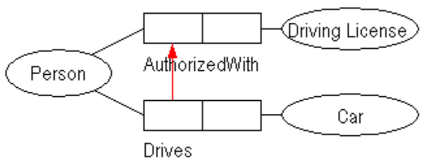


Fig. 6.17. Example of ORM Subset constraint.

The complete verbalization templates of all ORM constraints are illustrated with examples from each of the five translated languages and presented in appendix B3.

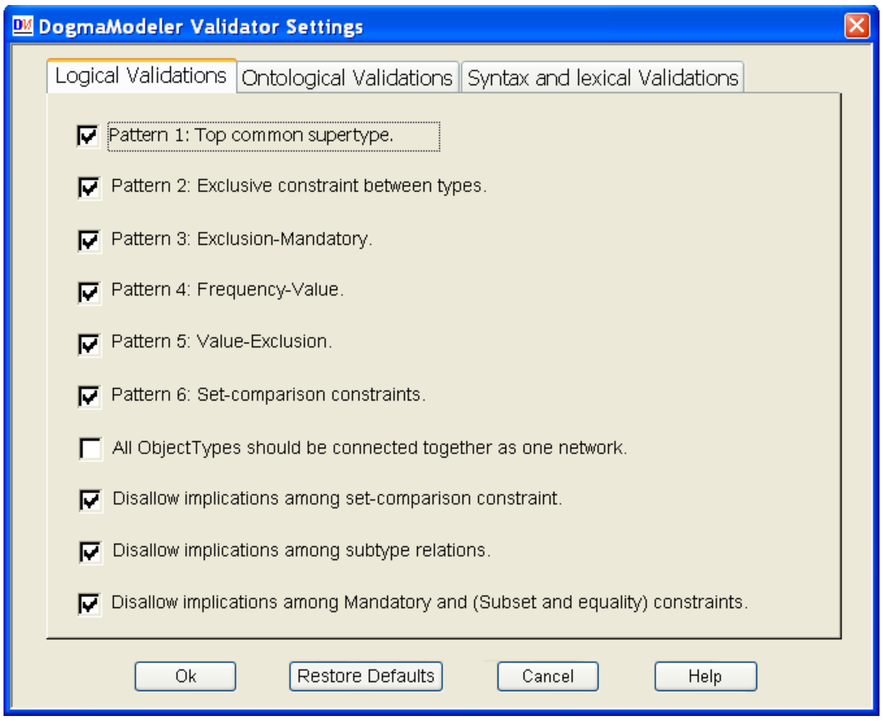
Notice that the verbalization templates (which are typically attached with the DogmaModeler as "setting-files") are not intended to be customized by "normal" ontology builders. Rather, the idea is to equip ontology engineers and experts with an easy to translate (or improve) verbalization mechanism.

6.4 Validation of application axiomatization[89]

[89] Later, this work was revised and extended ([JS06] and [JH08]). In addition, DogmaModeler was extended to allow description logic based reasoning (See [J07], [J07b], and [JD06]).

DogmaModeler supports various types of validations. These are logical validations, ontological validations as well as syntax and lexical validations.

Logical validations typically are “satisfiability” and “implication reasoning” validations, which can be used to validate application axiomatizations. Fig. 6.18 displays these patterns as a menu in the DogmaModeler Validator Settings window. Users can choose to enable or disable the enforcement of these validation patterns when reasoning about the satisfiability of an application axiomatization. The DogmaModeler typically implements the algorithms of the satisfiability patterns that we have developed in section 4.5. The specification of the last three implication patterns is adopted from [H89].


DogmaModeler Validator Settings

Logical Validations | Ontological Validations | Syntax and lexical Validations

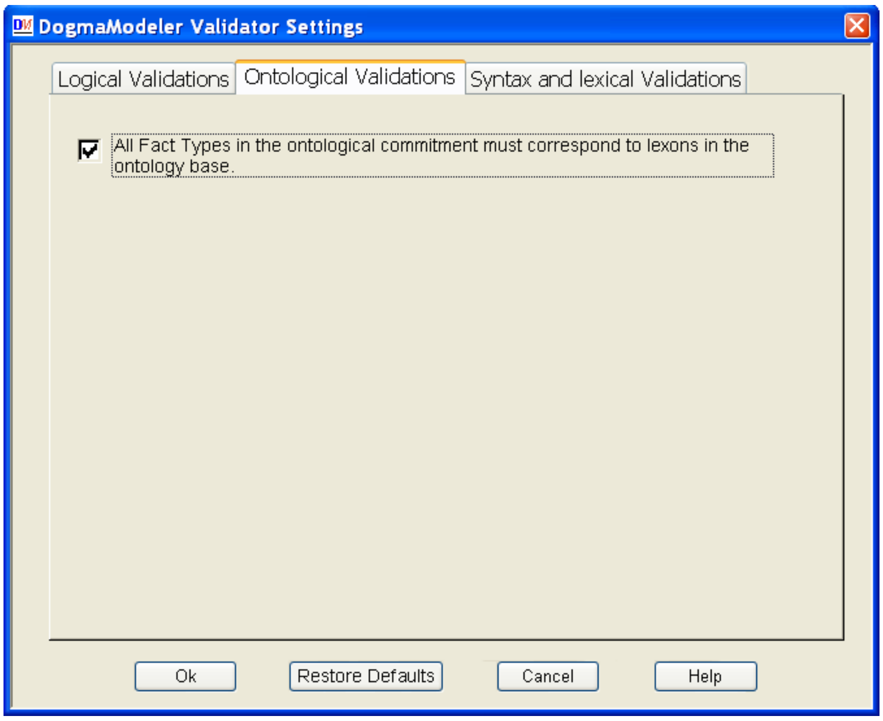
- [✓] Pattern 1: Top common supertype.
- [✓] Pattern 2: Exclusive constraint between types.
- [✓] Pattern 3: Exclusion-Mandatory.
- [✓] Pattern 4: Frequency-Value.
- [✓] Pattern 5: Value-Exclusion.
- [✓] Pattern 6: Set-comparison constraints.
- [ ] All ObjectTypes should be connected together as one network.
- [✓] Disallow implications among set-comparison constraint.
- [✓] Disallow implications among subtype relations.
- [✓] Disallow implications among Mandatory and (Subset and equality) constraints.

Fig. 6.18. DogmaModeler’s support of Logical validations.

-D

Ontological validation is concerned with ensuring that all fact-types in a commitment correspond to lexons in a given ontology base. See fig. 6.19.

If an application axiomatization is developed using DogmaModeler, the result of this validation is always positive, as users are unable to introduce new terminologies or fact-types unless they are defined in the ontology base. This validation is important in case application axiomatizations are modeled or modified using other tools.

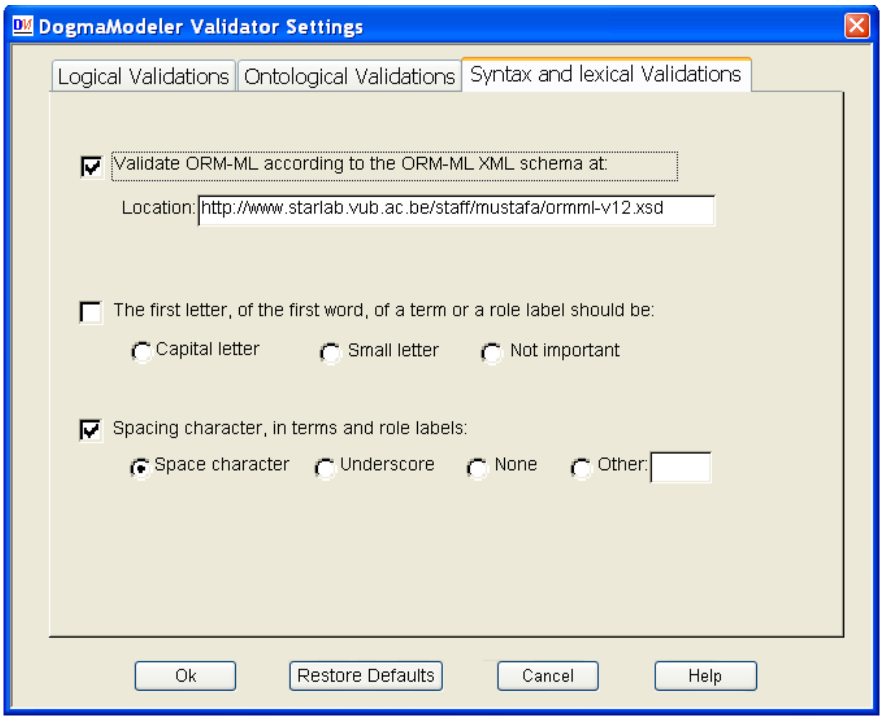


☑ All Fact Types in the ontological commitment must correspond to lexons in the ontology base.

Fig. 6.19. DogmaModeler's support of ontological validations.

Fig. 6.20 shows the menu of the syntax and lexical validations. As is apparent from the figure, these validation patterns are concerned with issues relating to grammar and formatting.

-D


DogmaModeler Validator Settings

Logical Validations | Ontological Validations | Syntax and lexical Validations

[✓] Validate ORM-ML according to the ORM-ML XML schema at:

Location: http://www.starlab.vub.ac.be/staff/mustafa/ormml-v12.xsd

[ ] The first letter, of the first word, of a term or a role label should be:

Capital letter | Small letter | Not important

[✓] Spacing character, in terms and role labels:

[✓ Space character] | Underscore | None | Other:

Fig. 6.20. DogmaModeler's support of syntax and lexical validations.

Outlook: Validations at the ontology base level are not yet supported in the DogmaModeler. This topic is considered in an upcoming paper related to this thesis (see section 8.3). Validations at the ontology base level should include, mainly the ontological quality and precision of an axiomatization. One example is how precisely a given set of lexons capture all aspects of the intended meaning of the ontology vocabularies and nothing else (i.e. all and only the intended meaning). As we have discussed in section 3.3.7 (and illustrated by examples), systematic quality and precision at the ontology base level can be achieved by incorporating primitives of upper level or foundational ontologies. Furthermore, some lessons on how to validate and deal with the lexical issues of the ontology vocabulary can be learned from the "lexical semantics" research

-D

community[90], such as, the use of nouns and adjectives verses terms, verbs verses roles, the modeling of idioms, the specific uses of metaphors, singulars, plurals, etc.

6.5 Axiomatization libraries

In this section, we present the DogmaModeler's support of axiomatization libraries.

As the number of axiomatizations is expected to grow rapidly, developing axiomatization library systems is a recognized need [DF01] [SGG+05]. The main goals of such libraries are to facilitate the reusability, organization, and management of axiomatizations. Metadata is the key infrastructure that enables the development of such axiomatization libraries [SGG+05]. Metadata is a systematic method (used by both human and machines) for describing axiomatization resources. It provides potential users of an axiomatization with basic knowledge of this resource.

A metadata record generally consists of a set of pre-defined elements that describe a resource (sometimes called tags, or attributes) and each element can have one or more values.

The DogmaModeler allows different metadata standards (e.g. Dublin-Core, LOM, etc.) to be used for describing axiomatizations. However, as such metadata standards are very general in their description of resources and not concerned with describing ontological resources in particular, we have developed a set of metadata elements as an extension to (and specialization of) the Dublin-Core metadata standard. Our metadata elements are intended to describe ontological resources. Further, by extending a common standard (i.e. Dublin-Core) we aim to gain more

[90] Specially from the emerging WordNet-alike (or so called "mental lexicons") communities, such as http://www.globalwordnet.org/ (January, 2005).

adoptability of our elements and compatibility with legacy resources and systems.

In fig. 6.21, we present an ORM representation of our metadata elements. This specification is used in the DogmaModeler as a meta-model of the axiomatization library. For the sake of brevity, the definitions of these elements (i.e. metadata glossary) are presented in appendix B1. Section 5.2.1 shows how these elements can be used in ORM-ML.

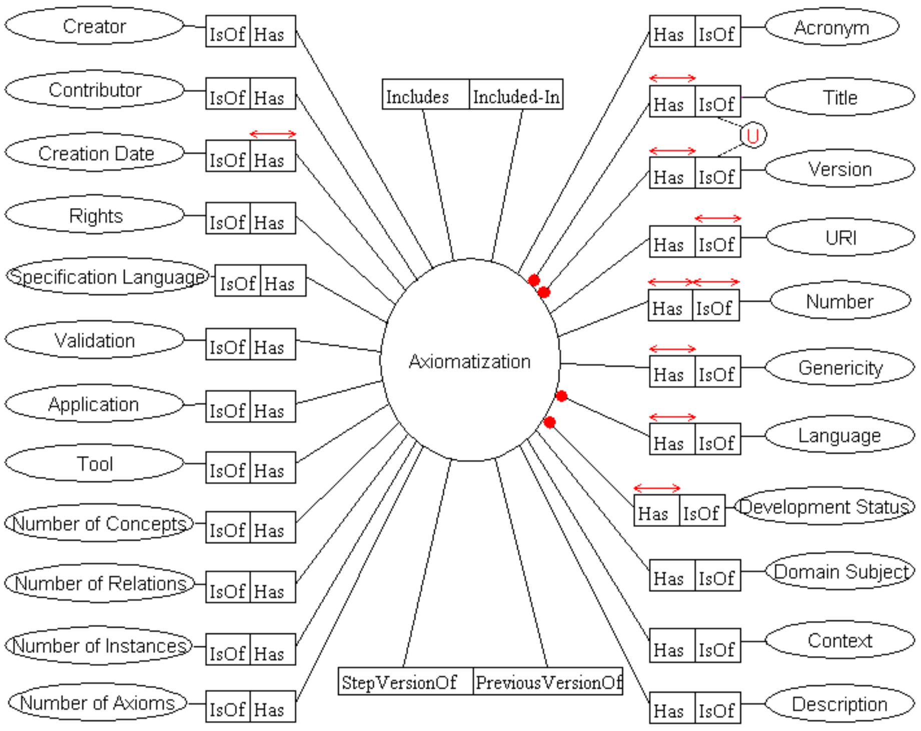
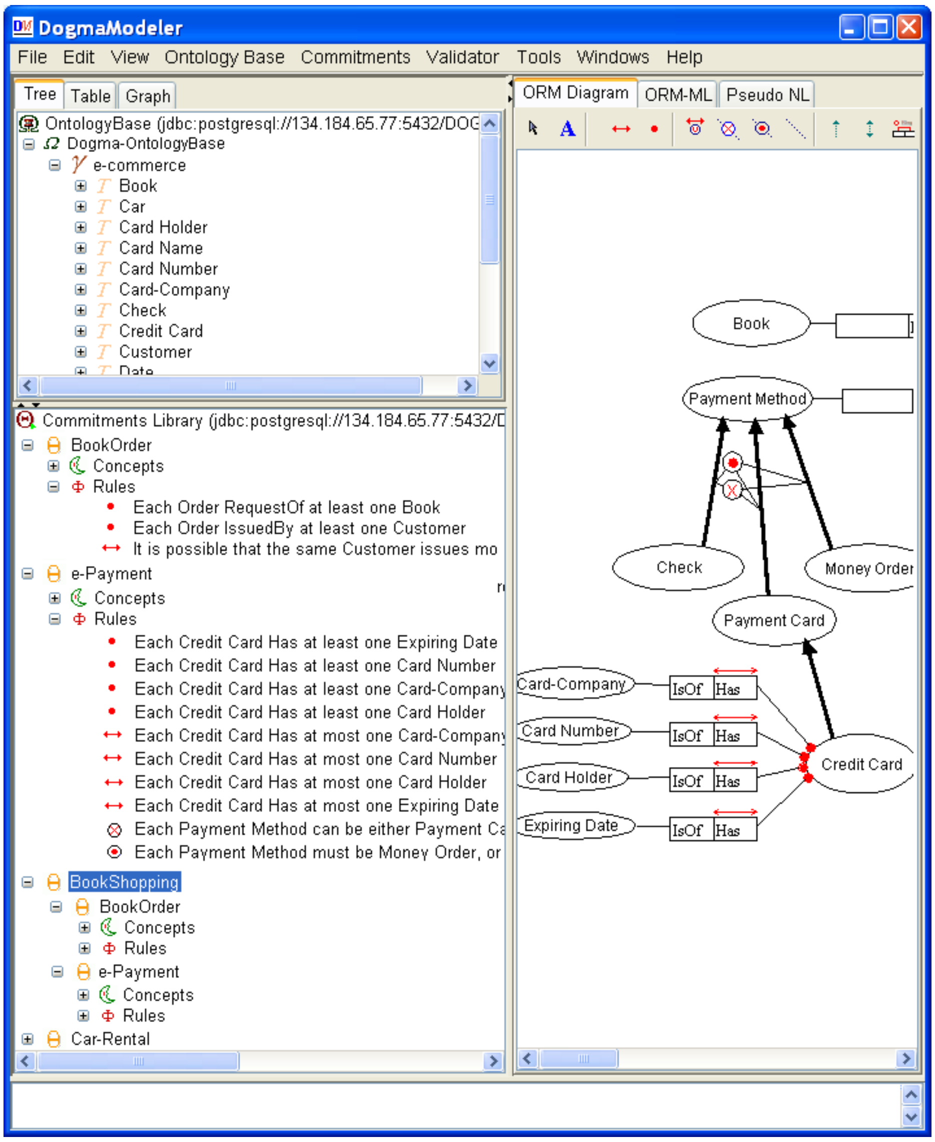


Fig. 6.21. DogmaModeler's a meta-model of the axiomatization library.

In fig. 6.22, we show the commitment library widow. In this window, DogmaModeler users can add, delete, manage, and brows application axiomatizations. Notice that an axiomatization may include other axiomatizations. For example, the "BookShopping" axiomatization is a

-D

composition of the "BookOrder" and the "e-Payment" axiomatizations[91]. Such an axiomatization is called a modularized axiomatization (see section 4.4.3).



```
OntologyBase (jdbc:postgresql://134.184.65.77:5432/DOG
  Dogma-OntologyBase
    e-commerce
      Book
      Car
      Card Holder
      Card Name
      Card Number
      Card-Company
      Check
      Credit Card
      Customer
      Date

Commitments Library (jdbc:postgresql://134.184.65.77:5432/D
  BookOrder
    Concepts
    Rules
      • Each Order RequestOf at least one Book
      • Each Order IssuedBy at least one Customer
      ↔ It is possible that the same Customer issues mo
  e-Payment
    Concepts
    Rules
      • Each Credit Card Has at least one Expiring Date
      • Each Credit Card Has at least one Card Number
      • Each Credit Card Has at least one Card-Company
      • Each Credit Card Has at least one Card Holder
      ↔ Each Credit Card Has at most one Card-Company
      ↔ Each Credit Card Has at most one Card Number
      ↔ Each Credit Card Has at most one Card Holder
      ↔ Each Credit Card Has at most one Expiring Date
      ⊗ Each Payment Method can be either Payment Ca
      ⊙ Each Payment Method must be Money Order, or
  BookShopping
    BookOrder
      Concepts
      Rules
    e-Payment
      Concepts
      Rules
  Car-Rental
```

Fig. 6.22. DogmaModeler's support of axiomatization libraries.

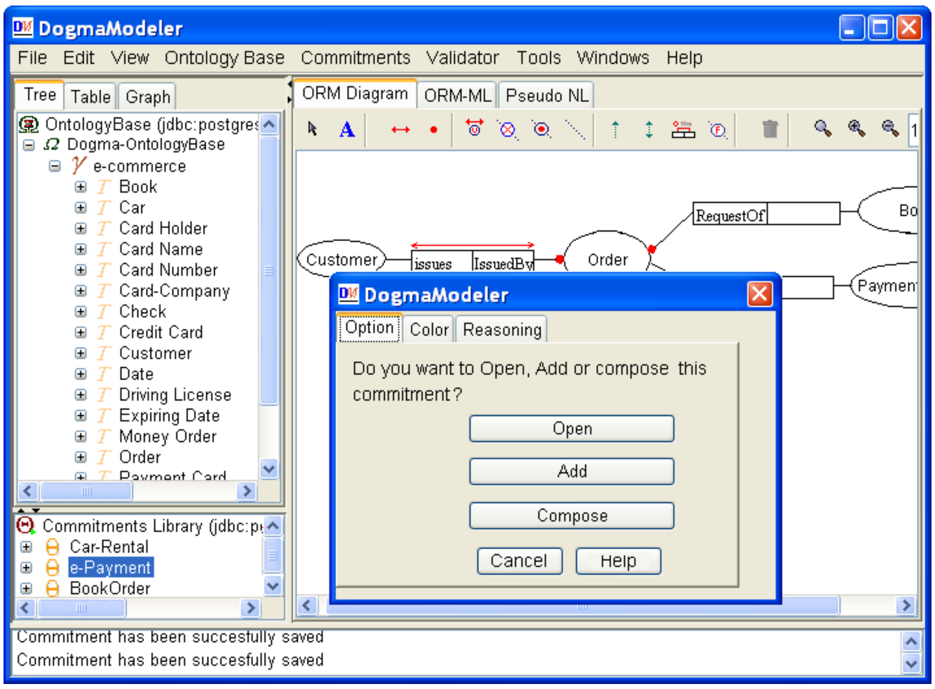
[91] See fig. 4.2.

-D

6.6 Composition of axiomatization modules

The DogmaModeler supports the automatic composition of axiomatization modules. It typically implements the composition algorithm we presented in chapter 4.

When dragging or dropping an axiomatization from the commitment library window to the commitment modeling window, a menu appears asking the user whether he/she want to Open, Add, or Compose this commitment (see fig. 6.23.).



Do you want to Open, Add or compose this commitment?

Commitment has been succesfully saved
Commitment has been succesfully saved

Fig. 6.23. DogmaModeler's support of axiomatization libraries.

The "Add" choice is merely a copy-paste operation that copies all lexons and constraints to the axiomatization that is being edited in the commitment-modeling window. No reasoning steps are attached or associated with the Add operation.

When choosing “Compose”, the DogmaModeler composes the “dragged axiomatization” with the “opened axiomatization(s)” in the modeling window. During this composition, the DogmaModeler implements the composition algorithm and the associated reasoning steps that we specified in chapter 4. If the result cannot be satisfied, the composition is considered an incompatible operation and thus *terminated*.

To facilitate simplicity in the viewing and editing of a modular axiomatization, the DogmaModeler allows users to draw each module in a different color. Users are also prevented from modifying any of the composed modules. In other words, users cannot delete or change any fact-types or constraints that originate from any of the composed axiomatizations.

When generating the ORM-ML (of a modular axiomatization), the DogmaModeler allows the users to choose to either 1) refer to the axiomatizations composed by their URIs, or 2) include the content of these composed axiomatizations (as sub-commitment) inside the ORM-ML document. Fig. 6.24 illustrates the ORM-ML representation of a modular axiomatization using RUIs as references to the composed modules. In this way, each of the composed modules will be fetched when opining (or using) the modular axiomatization. The main disadvantage of this method is that any changes to the modules may influence the satisfiability of the composition.

```
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.starlab.vub.ac.be/staff/mustafa/ormml.xsd'
xmlns:dc='http://purl.org/dc/elements/1.1/' OntologyBase="Dogma OntologyBase" Context="e-
commerce">
<ORMMeta> <Meta name="title" content="BookShoping"/> … </ORMMeta>
<ORMBody>
    <Subcommitment URI="http://www.starlab.ac.be/staff/mustafa/bookorder.ormml"/>
    <Subcommitment URI="http://www.starlab.ac.be/staff/mustafa/e-payment.ormml"/>
</ORMBody>
</ORMSchema>
```

Fig. 6.24. An example of the ORM-ML representation of a modular axiomatization, using URIs.

In the second choice, users can choose to include “a copy” of each module as a subpart of the ORM-ML document (see fig. 6.25.). In this way, several problematical issues are prevented, such as the influence of module changes and broken links. However, the main disadvantage of this method is that some useful changes, to the original modules, will not be captured.

The DogmaModeler allows users to decide on the most appropriate method, given their application scenario, the steadiness of their module evolution and whether their usage is on or off-line etc.

```
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.starlab.vub.ac.be/staff/mustafa/ormml.xsd'
xmlns:dc='http://purl.org/dc/elements/1.1/' OntologyBase="Dogma OntologyBase" Context="e-
commerce">
<ORMMeta> <Meta name="title" content="BookShoping"/> .... </ORMMeta>
<ORMBody>
    <Subcommitment >
        <ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:noNamespaceSchemaLocation='http://www.starlab.vub.ac.be/staff/mustafa/ormml.xs
        d' xmlns:dc='http://purl.org/dc/elements/1.1/' OntologyBase=" Dogma OntologyBase "
        Context="e-commerce">
            <ORMMeta><Meta name="title" content="BookOrder"/></ORMMeta>
            <ORMBody> .... </ORMBody>
        </ORMSchema>
    </Subcommitment>
    <Subcommitment >
        <ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:noNamespaceSchemaLocation='http://www.starlab.vub.ac.be/staff/mustafa/ormml.xs
        d' xmlns:dc='http://purl.org/dc/elements/1.1/' OntologyBase=" Dogma OntologyBase "
        Context="e-commerce">
            <ORMMeta><Meta name="title" content="e-Payment"/></ORMMeta>
            <ORMBody> .... </ORMBody>
        </ORMSchema>
    </Subcommitment>
</ORMBody>
</ORMSchema>
```

Fig. 6.25. An example of an ORM-ML representation of a modular axiomatization, where the content of a module is included as a sub-commitment.

6.7 Other functionalities

This section briefly touches on a few other functionalities of DogmaModeler, *for example, those that deal with* ontology-driven forms, and ontology multilingualism.

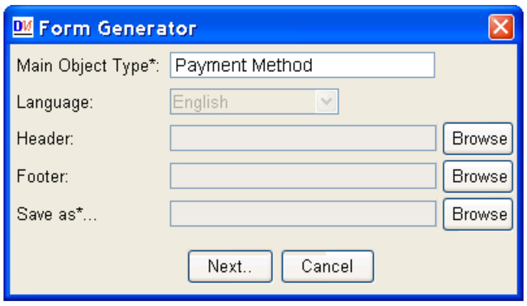
6.7.1 Ontology-driven forms

DogmaModeler supports the automatic generation of a web form based on a given ORM-ML axiomatization. This functionality first generates an XForm[92] from the given axiomatization, before generating a HTML-form out of the generated XForm. The purpose of generating an intermediate XForm is to allow changes to the layout of a form before generating the HTML-form. This functionality has been successfully used in the CCFORM thematic-network project for generating customer complaint web-forms based on the CContology (see chapter 7).

In the following paragraphs, we present this functionality at the abstract level. For more details, please refer to [JLVM03].

To map an ORM schema into Xform, users should first select the main Object-Type that they want to build a form about (see fig. 6.26.). DogmaModeler then maps the ORM schema into a hierarchal structure based on the previously selected Object-Type that functions *as a root*. For example, fig. 6.27 shows the generated hierarchy of the e-Payment axiomatization that we have presented in fig 4.2.

[92] The classical design of Web forms does not separate the purpose of a form from its layout. Conversely, Xforms are comprised of separate sections that describe what the form does, and how it looks. XForms are considered the next generation of web forms but XForm technology is still a work in progress and is not yet standardized. In the DogmaModeler, we use the NanoWorks XForm XML form specification (webpage http://xform.nanoworks.org , January 2005). Some of the preferable features of NanoWorks XForm are that (1) it generates standard HTML and javascript that works with any browser, (2) it is open source and requires no special plug-ins (3) it significantly reduces the coding necessary to build and maintain complex form interfaces, (4) it insures data integrity by validating user input on the client-side and the server-side, (5) it reduces the likelihood of error by encapsulating form structure and validation, and (6) it *creates a record of user data as an XML document*.



Main Object Type*:	Payment Method
Language:	English
Header:	
Footer:	
Save as*...	

Fig. 6.26. The step of generating an ontology-based web form.

Before generating the Xform specification, the "Xform Tree" window in fig. 6.27 enables users to delete the unwanted nodes (so they do not appear in the generated form), and to sort the nodes according to a desired order (in the form).

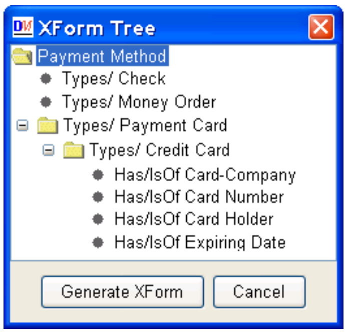


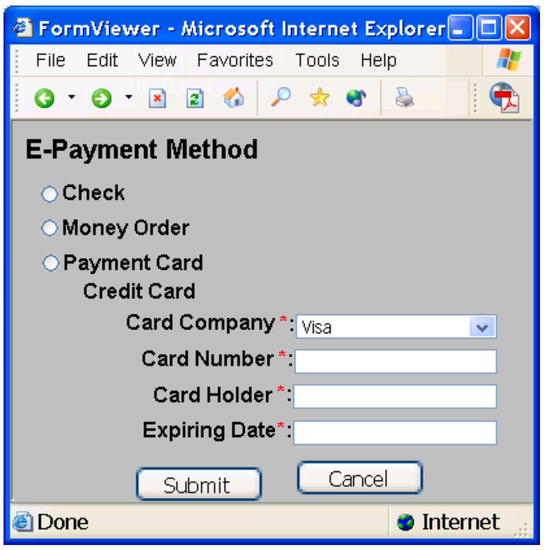
Fig. 6.27. the "Xform Tree" window.

We have adopted the approach presented in [EWHLF02] for mapping an ORM schema into a hierarchy, and for eliminating the possible cycles in the schema. This approach is used for generating an XML-scheme out of a given EER diagram.

In the last step, the DogmaModeler maps the generated hierarchy into the Xform specification which then can be directly mapped into an HTML specification using a NanoWorks web server.

In fig. 6.28 we show the resultant web form from the above example. For the sake of brevity, the Xfrom specification is not presented here.

-D


FormViewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Internet

E-Payment Method

- ○ Check
- ○ Money Order
- ○ Payment Card

Credit Card

Card Company *:	Visa
Card Number *:	
Card Holder *:	
Expiring Date*:	

Submit Cancel

Fig. 6.28. The resultant web form of e-Payment axiomatization.

HTML does not allow for the encoding of all ORM constraints at the client-side (e.g. to apply integrity constraints when populating a web form). The NanoWorks server however, does allow the other constraints to eforced at the server-side. In the DogmaModeler, mandatory constraints are mapped into (and so can be enforced) using JavaScript at the client-side. A value constraint is mapped into the “Select” HTML element. In case the value constraint is not companied with an internal uniqueness, then the “Multiple” HTML attribute is added. Depending on the companion of the totality and exclusive constraints, subtypes are mapped into radio buttons or check boxes. See [JLVM03] for more details and examples.

6.7.2 Ontology Multilingualism

The DogmaModeler supports the multilingual lexicalization of ontologies. Given an ontology base (lexicalized in a certain language, called the ontology *native* language), the DogmaModeler allows ontology builders to build a list of one-to-one translations into other languages. This list is

-D

not seen as part of the ontology itself. Rather, it belongs to a certain application scenario or a group of users.

We postpone the discussion on this issue and its DogmaModeler's support to section 7.4. We shall illustrate our approach to multilingual lexicalization of the CContology, discuss multilingual ontologies verses multilingual lexicalization of ontologies, and provide some methodological guidelines on the translation of ontology terms.

6.8 Discussion and conclusions

In this chapter, we have presented the DogmaModeler, our prototype ontology engineering tool. We have shown how to model and represent both domain and application axiomatizations. We have shown also how existing lexical resources can be incorporated in concept/gloss modeling. The adoption of conceptual data modeling techniques for ontology engineering is illustrated through the use of ORM as a modeling and specification language of application axiomatizations. We have presented an easy to customize verbalization template that allows non-ontology experts to (help) check, validate, or even build application axiomatizations. DogmaModeler supports and implements the automatic composition of modules as well as the representation and validation of modular axiomatizations. A set of carefully defined ontology metadata is proposed to enable the implementation of axiomatization libraries.

Although the DogmaModeler introduced in this chapter is a prototype, it has been successfully applied in a number of real-life and large projects such as CCFORM, FFPOIROT, SCOP, etc. It has been acknowledged as an intuitive tool for non-ontology experts, particularly because of the graphical and verbalization support it provides. In the next chapter, we proceed to report our experiences and main achievements in using the tool in the CCFORM project, specifically, for developing a Customer Complaint ontology (CContology).

Acknowledgement: I would like to express my sincere gratitude to all my master’s students and colleagues who have helped me in the implementation of DogmaModeler. Particularly, I’d like to thank Andriy Lisovoy who programmed the main architecture and components of the tool, Jan Demey for his help in programming some components of the early version, Hai Nguyen Hoang who helped me in the implementation of the verbalization functionality, and Quoc Hung for implementing WordNet mapping to DogmaModeler. I am also in debt to all colleagues and project partners who so generously shared their comments and made suggestions to improve the tool.

Chapter 7

The CCFORM Case Study

"If customers do not hesitate to use on-line service, it will facilitate their day to day life. The development of electronic commerce must not be limited to a group of people and to an experimental stage. It can, for instance, become a huge facility for house bound citizens, such as mothers with small children, or handicapped persons…"

(The CCFORM Project)

**Later, this chapter was revised and published in [J08]*

In this chapter, we outline our experience in applying the methodological principles and the tool for developing a Customer Complaint ontology (CContology). This ontology has been developed within the EU CCFORM thematic-network project[93] which is introduced in section 7.1. The CContology itself is presented in section 7.2, while section 7.3 provides a discussion of the application and the lessons learned in the process. A methodology for multilingual lexicalization of ontologies is presented in section 7.4 before conclusions are drawn in section 7.4.

[93] (IST-2001-34908), 5th framework.

7.1 Introduction

The use of the Internet for cross-border business is growing rapidly. However, in many cases, the benefits of electronic commerce is not exploited fully by customers because of the frequent lack of trust and confidence in online cross-border purchases. To achieve fair trading and transparency in commercial communications and transactions, effective cross-border complaint platforms need to be established and involved in e-business activities [CIHF02] [CW87] [ABA02].

The CCFORM project aims to study and *reach a consensus* about the foundation of online customer complaint mechanisms by developing a standard but extensible form (called CC-form[94]) which has widespread industry and customer support. This CC-form must facilitate cross-language communication to support cross-border e-commerce and should be easy to implement in software tools. The CC-form will raise the basic standard of complaints management, and should be extended in vertical markets to provide sector-wide solutions to allow service providers to gain competitive advantages.

There are several challenges involved in establishing and standardizing such a CC-form: (1) Legal bases: the sensitivity of cross-border business regulations and privacy issues. (2) The diversity of language and culture: controlling and standardizing the semantics of the complaint terminology so that the intended meaning of the term gets across, even in the different languages. (3) Consumer sensitivity and business perspectives. (4) Extensibility: the flexibility of extending the CC-form (perhaps dynamically) according to market needs and standards. This would mean for example, extending the kinds of problems that a complainant can

[94] We refer to the project as CCFORM and to a customer complaint form as " CC-form". One may imagine a CC-form as one page web-form, or several pages that can be filled in several steprs.

complain about and extending the kinds of resolutions, managing who may extend what, etc.

In order to tackle such challenges and to perfect the reference model for the complaint form, the major work in the CCFORM project has been divided into six topic panels (TPs), each consisting of 10-15 specialized members. Each panel has been intensively discussing different issues: TP1- Legal Affairs, TP2 - Consumer Affairs, TP4 - Standards for SMEs, TP5 -Alternative Dispute Resolution Systems, TP6 - Ontology and Extensibility, TP7 - Vertical markets.

This work outlines our main achievements in the "Ontology and extensibility, including multilingual and cultural issues" topic panel. The goal of this topic panel, TP6, is to undertake extensibility and multilingual demands. To approach this, a customer complaint ontology (CContology) has been developed and lexicalized in multiple languages.

7.2. Customer Complaint ontology

The *customer complaint ontology* (CContology) intends to capture the main concepts in the "customer complaint management" domain. Its core covers a semantic description of complaints that could be issued by any legal person against any other legal person (NGO, company, natural person, etc.). The CContology comprises classifications of complaint problems, complaint resolutions, complainant, complaint-recipient, "best-practices", rules of complaint, etc.

The main intended impact of the CCFORM project is the future initiation of a European online complaint platform that will provide a trusted portal between consumers and business entities. In this respect, the ontology is intended to become the basis for a future *core ontology* in the domain of customer complaint management (for both humans and machines). Applying the CContology in such an European online complaint platform will facilitate further refinements of the CContology.

The main uses of such an ontology are 1) to enable consistent implementation (and interoperation) of all software complaint management mechanisms based on a shared background vocabulary, which can be used by many stakeholders. 2) to play the role of a *domain ontology* that encompasses the core complaining elements and that *can be extended by either individuals or groups of firms*; and 3) to generate CC-forms based on its ontological commitments and to enforce the validity (and/or integrity) of their population.

Although this CContology has been developed and reviewed by six topic panels, in its current state, it can only be considered a proposal. The CCFORM community is representative of a sizable cross-section of the domain but is not a standardization body. Nor is it in the position to insist on a *de facto* enforcement of this ontology as a generally agreed semantic specification. However, the approach presented in this paper is designed to initiate and drive such a process.

The CContology consists of a domain axiomatization (i.e. the ontology base that represents the lexons and the term glossary) and seven application axiomatization modules: Complaint Problems, Complaint Resolutions, Complaint, Complainant, Complaint-Recipient, Address, and Contract.

7.2.1 Customer-complaint domain axiomatization

This axiomatization consists of about 220 concepts and 300 lexons, which characterize the core concepts in the customer-complaint domain. The three representation units of this domain axiomatization (i.e. the ontology base) are: context, terms and their glosses, and the set of lexons.

“Customer Complaint” Context

As we have discussed in section 3.3.5 and in section 6.2.1, context is the first building block for developing a domain axiomatization. It plays a scoping role, through which the interpretation of the intended meaning of the ontology terminology is bounded.

In the CContology, the "Content ID" is called the "*Customer Complaint*" context, or the *CCcontext in short*. The "Context Description" is defined as follows:

Background knowledge (i.e. explicit, implicit, or tacit assumptions) about all (activities, communications, institutions, people, places, objects, etc.) that are involved in consumer-provider relationships, regarding contractual and non-contractual complaining issues.

These assumptions can be understood (i.e. can be found explicitly or intuitively) in the following sources:

- *European Distance Selling Directive (97/7/EC), on the promotion of consumers in respect of distance contracts.*
- *European e-Commerce Directive (2000/31/EC) on certain legal aspects of information society services, in particular, electronic commerce, in the Internal Market*
- *European Data Protection Directives (95/46/EC and* 97/66/EC*) on the protection of individuals with regards to the processing of personal data and on the free movement of such data.*
- *European Directive (99/44/EC) on aspects of the sale of consumer goods and associated guarantees.*
- *European Directive (98/27EC) on Injunctions for the Protection of Consumers' Interests.*
- *CEN/TC331 Postal Services EN 14012:2002 Quality of*

-D

Service – Measurement of complaints and redress procedures.

- *"Best practice" guidelines, The Nordic Consumer Ombudsmen's position paper on trading and marketing on the Internet and other similar communication systems(http://econfidence.jrc.it, June 2002)*
- *CCFORM Annex 1, (IST-2001-34908, 5th framework).*
- *CCFORM Report On Copyright And Privacy Recommendations (Deliverable D.5.3).*
- *CCFORM user guide and business complaints (Deliverable D.5.1.1).*
- *CCFORM Company user guide (Deliverable D.5.1.2).*
- *CCFORM Web publication of CCform User Guides in 11 languages (Deliverable D6.11).*
- *Code of Conduct (CCFORM deliverable).*

Remark: For the sake of brevity, many resources (regulations at the European and national levels, best practices, existing online complaining (plat)forms, etc.) are not mentioned here. However, references to these resources can be found inside the resources listed above.

We have learned during the definition process of the above CCcontext that it is not an easy task, and it cannot be defined rigidly in the early phases of the development of the CContology. As none of our team was an ontology expert, we provided a draft definition and investigated by providing many different examples of application scenarios that this context should cover[95]. For example, we have questioned whether the context should cover applications such as customer-relationship-management, market analyses, sales force automation and so forth; whether it should cover all consumer regulations in any country or only in Europe; whether it should cover all commercial activity, in any place and at any time; which documents, laws and regulations should be our main references, etc. Such questions led not only to the CCcontext definition (which was achieved after several iterations), but also propelled the team to discuss deeply and even redefine the scope of the CCFORM goals.

Vocabularies and their glosses

Within the "Customer Complaint" context, we define 220 terms. These terms and their glosses (Called CCglossary) are provided in appendix C1.

The CCglossary was developed (and reviewed) over several iterations. The first iteration was accomplished by a few (selected) experts before the lexon modeling process was started. Further iterations have been carried out in parallel with the lexon modeling process. The final draft was reviewed and approved by several topic panels. It is probably worth noting that intensive discussions were carried out (by legal experts, market experts, application-oriented experts) for almost every gloss. We have found that the gloss modeling process is a great mechanism for brainstorming, domain analyses, domain understanding and for reaching (and documenting) consensus. Furthermore, it allowed non-ontology experts to participate actively in the ontology modeling process[96].

As shall be discussed in section 7.4, this CCglossary, which has been developed in English, has played the role of *the key* reference for lexicalizing the CContology into 11 other European languages.

[95] This investigation was done to prevent the CContology from being dependent on the CC-form application scenario which the team had in mind during the early phases.
[96] Some CCFORM partners have noted that the CCglossary is the most useful component in the CContology.

Translators have acknowledged that it guided their understanding of the intended meanings of the terms and allowed them to achieve better translation quality.

Lexons

Stemming from the 220 terms within the “Customer Complaint” context, we have developed 300 lexons, which can be found in appendix C2. Most of these lexons represent taxonomies of complaint problems, complaint resolutions, complainant, complaint recipient, etc.

The first draft of the lexons has been developed based on presentations and discussions between the members of Topic Panel 6 (Ontology and Extensibility). One of the most important inputs, for the first draft, was the complaint categorization survey [VS03] that was performed by two of the panel members. Further, refinements and investigations were performed during meetings and workshops that we organized in cooperation with other topic panels.

7.2.2 Customer-complaint application axiomatization

Given the previously presented “customer complaint” domain axiomatization, seven application axiomatization modules have been developed. The intended meaning of the terminology used in these application axiomatization modules is restricted to the terminology defined at the domain axiomatization level.

The application axiomatization modules are intended to play the role of conceptual data schema(s) for CC-forms development. Any CC-form, including its population, should be based on (i.e. commit to) the CContology through those axiomatization modules. A CC-from can be constructed manually or generated automatically (as has been illustrated in section 6.7.1); nevertheless, the semantics of all elements in this CC-from (i.e. the data fields) should be defined in the CContology.

As stated earlier in this chapter, the seven application axiomatization modules are: Complaint problems, Complaint resolutions, Contract, Complaint, Complainant, Complaint Recipient, and Address. Depending on an application's usability requirements, these modules can be used individually or composed to form a modular axiomatization(s).

In the following section, we provide a brief description of each module, including its ORM graphical representation. The ORM-ML representation of all modules, and their verbalization into pseudo natural language, are presented in appendix C3.

Complaint Problems

Fig. 7.1 shows the "Complaint Problems" axiomatization module. It represents a taxonomy of complaint problems.

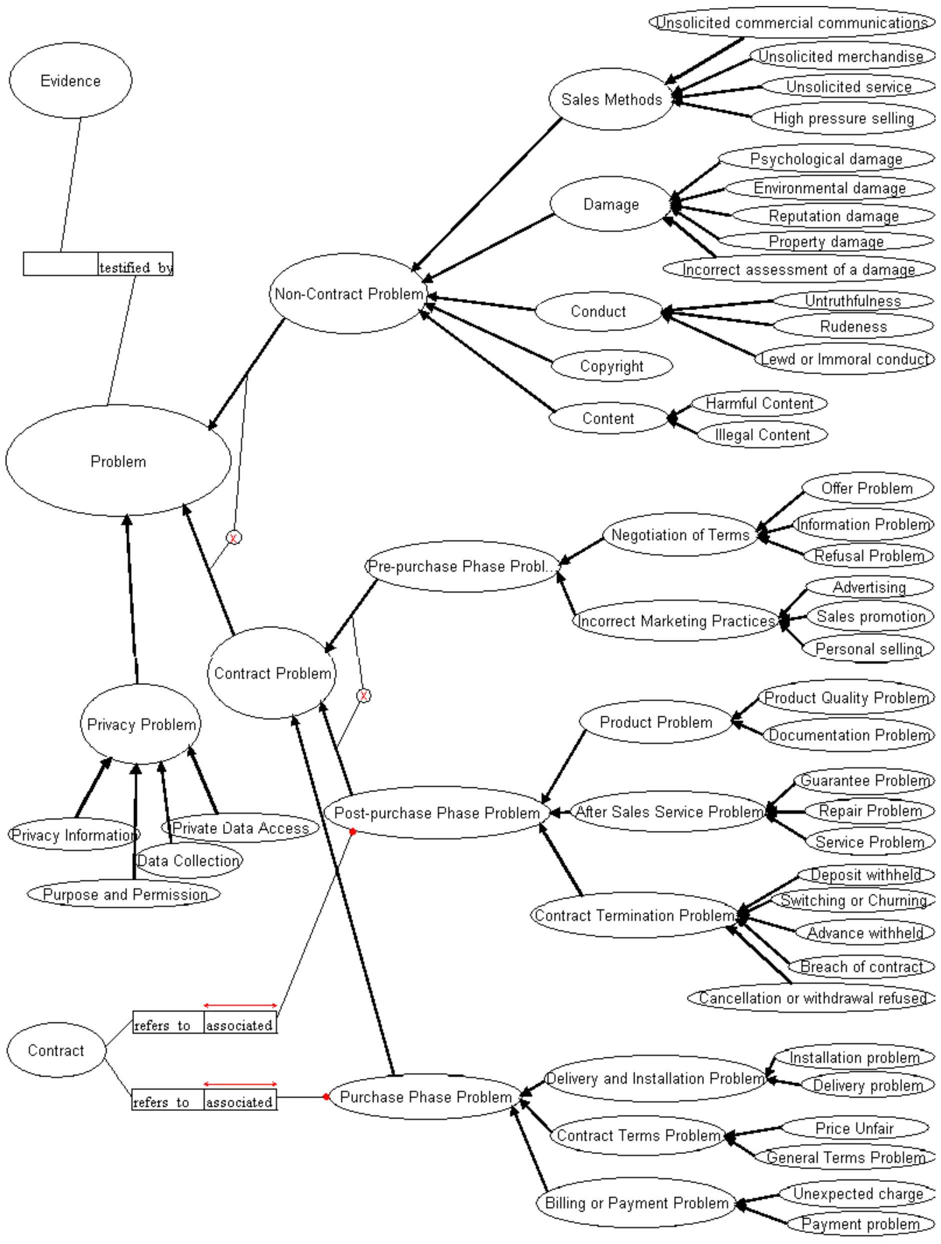


Fig. 7.1. The “Complaint Problems” application axiomatization module.

We distinguish between a ‘Complaint’ and a ‘Problem’. A ‘Complaint’ *describes* one or more ‘Problems’. While the concept ‘Problem’ is defined as “A source of difficulty or dissatisfaction”, the concept ‘Complaint’ is defined as “An expression of grievance or resentment issued by a complainant against a compliant-recipient, describing a problem(s) that needs to be resolved”.

Within the “customer complaint” domain, a ‘Problem’ can be a ‘Privacy Problem’, or either a ‘Contract Problem’ or a ‘Non-contract Problem’. A ‘Contract Problem’ can be a ‘Purchase Phase Problem’, or either a ‘Pre-purchase Phase Problem’ or a ‘Post-purchase Phase Problem’. It is mandatory for both ‘Purchase Phase Problems’ and ‘Post-purchase Phase Problems’ to be associated with a ‘Contract’. For any type of problem, ‘Evidence’ might be provided for investigation purposes.

Remark: In this “Complaint Problems” module, only four classification levels are presented, all of which are the popular categories in most CC-forms. Further classifications of complaint problems can be found at the ontology base level.

Complaint resolutions

Fig. 7.2 illustrates the “Complaint Resolution” module, which present a taxonomy of ‘Complaint Resolutions’. A ‘Complaint Resolution’ is defined the CCglossary as “A determination for settling or solving a complaint problem(s)”. It can be requested by a complainant or offered by a complaint-recipient. A ‘Complaint Resolution’ can be an ‘Economic Resolution’, a ‘Symbolic Resolution’, or an ‘Information Correction’.

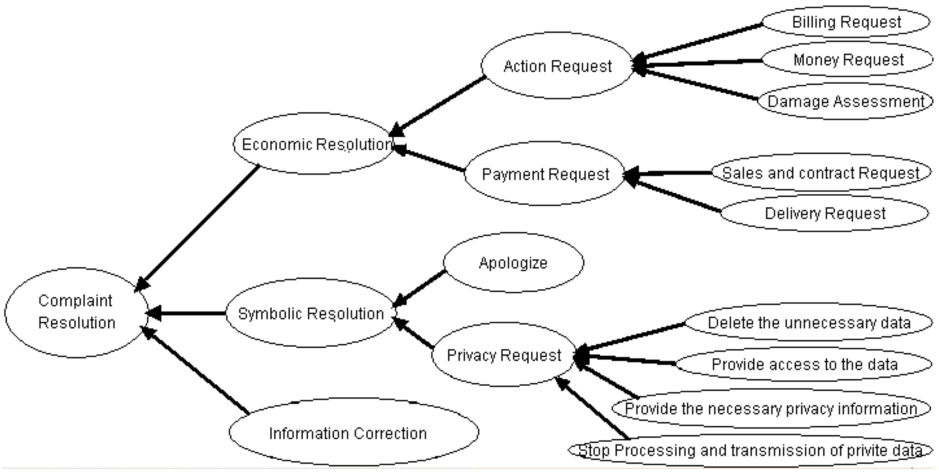


Fig. 7.2. The "Complaint Resolutions" application axiomatization module.

Contract

A 'Contract' is defined in the CCglossary as "a binding agreement, between two or more legal persons, that is enforceable by law". Under this definition, an invoice can also be a contract. Fig. 7.3 illustrates the "Contract" axiomatization module, which specifies the information that should be provided for a contract associated with a 'Purchase Phase Problem' or 'Post-purchase Phase Problem'. Notice that, for a CC-form, we speak of a 'Contract' from the moment there is a 'Contract Order Date'.

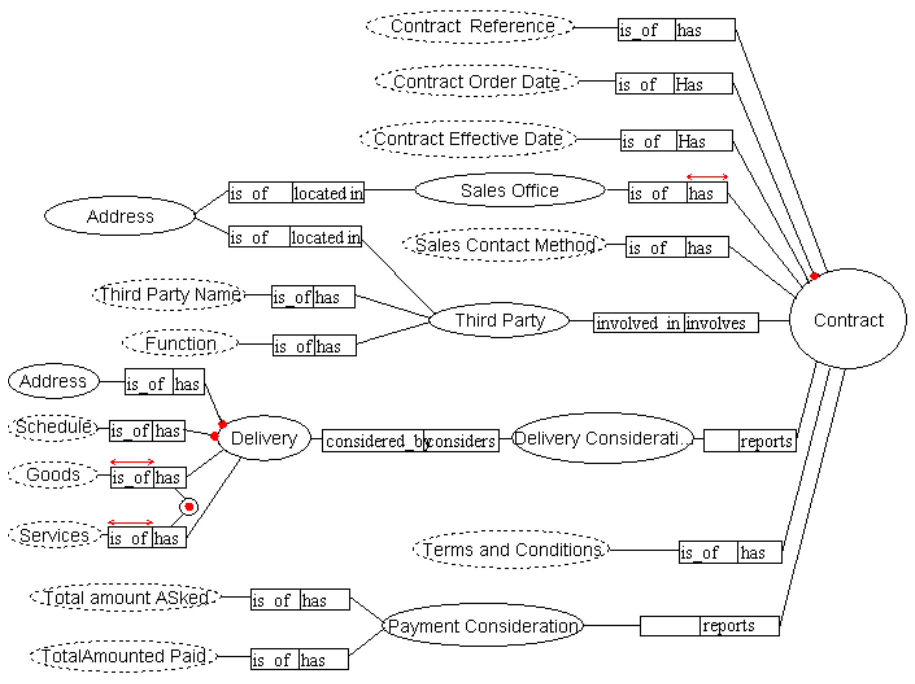


Fig. 7.3. The "Contract" axiomatization module.

Complaint

A 'Complaint' is defined in the CCglossary as "An expression of grievance or resentment issued by a complainant against a compliant-recipient, describing a problem(s) that needs to be resolved".

Fig. 7.4 illustrates the "Complaint" axiomatization module, which specifies the main concepts that can be associated with the concept 'Complaint'. A 'Complaint' must be issued by a 'Complainant' against a 'Complaint-Recipient', on a certain 'Date'. It must describe at least one 'Problem', and may request one or more 'Complaint Resolutions'. A 'Complaint' *might be* identified by a 'Complaint Number', which is typically used as a unique reference in a court or a complaint system.

-D

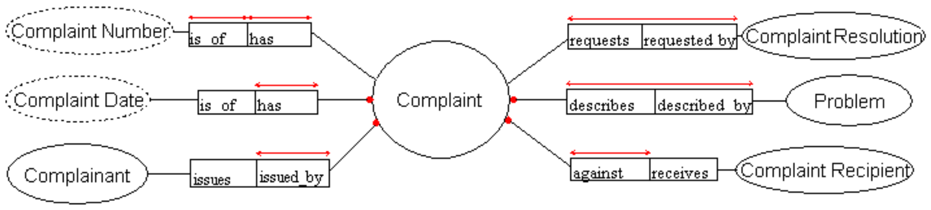


Fig. 7.4. The "Complaint" application axiomatization module.

Complainant

Fig. 7.5 illustrates the 'Complainant' axiomatization module. A 'Complainant' is defined in the CCglossary as "A legal person[97] who issues a complaint". In the customer complaint domain, and as commonly understood in most consumer regulations, a complainant must either be a 'Natural Person Complainant'[98] or a 'Non-Natural Person Complainant'[99], each implying a different legal basis for the handling of the complaint.

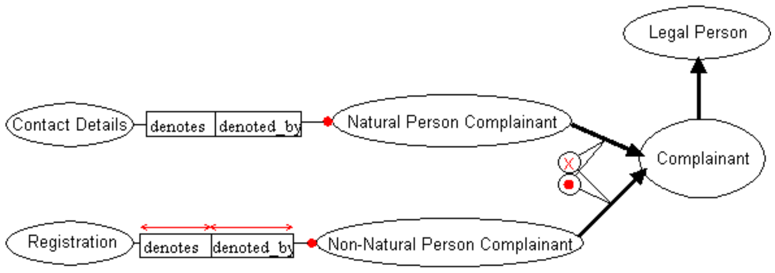


Fig. 7.5. The "Complainant" application axiomatization module.

The distinction between natural and non-natural person complainants is not only based on the variation of their complaint handling regulations,

[97] The concept 'Legal Person' is defined in the CCglossary as : "An entity with legal recognition in accordance with law. It has the legal capacity to represent its own interests in its own name, before a court of law, to obtain rights or obligations for itself, to impose binding obligations, or to grant privileges to others, for example as a plaintiff or as a defendant. A legal person exists wherever the law recognizes (as a matter of policy). This includes the personality of any entity, regardless of whether it is naturally considered to be a person. Recognized associations, relief agencies, committees, and companies are examples of legal persons".

[98] Such as a normal consumer.

[99] Such as a business customer.

but also on the legal preference (in any CC-from) for not obligating the inquiry of private information about the ‘Natural Person Complainant’, such as his/her ‘Name’, ‘Birth Date’, ‘Mailing Address’, ‘Religion’ etc.

Each ‘Natural Person Complainant’ must have ‘Contact Details’. The mandatory contact details are an ‘eMail’ and his/here ‘Country’ of residence. A ‘Non-Natural Person Complainant’ must be denoted by a certain ‘Registration’[100] that identifies him.

Complaint recipient

Fig. 7.6 illustrates the “Complaint Recipient” axiomatization module. A ‘Complaint Recipient’ is any legal Person to whom a complaint is addressed. Typically, when a ‘Complaint’ is issued against a ‘Complaint Recipient’, the ‘Contact Details’ *or* the ‘Registration’ of this ‘Complaint Recipient’ should be denoted[101].

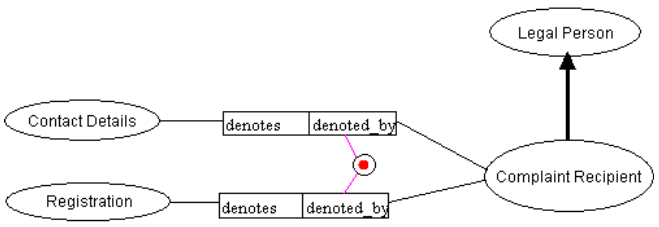


Fig. 7.6. The “Recipient” application axiomatization module.

Address

Fig. 7.7 illustrates the “Address” axiomatization module. The concept ‘Contact Details’, which is a channel of communication, is attributed by

[100] The concept ‘Registration’ is defined in the CCglossary as: “A certification, issued by an Administrative authority or an accredited registration agency, declaring the official enrollment of an entity. Typically, it includes the official name, mailing address, registration number, VAT number, legal bases, etc.”.

[101] Usually, all online customer complaint platforms provide a searchable database of many “Complaint Recipients”, which enables complainants to easily find the official names and addresses of ‘complaint recipients’

-D

both 'Name' and 'Address'. An 'Address'[102] must be either an 'Electronic Address' or a 'Mailing Address'. An 'electronic Address' can be either a 'Web Site', 'Telephone', 'eMail', or 'Fax'. A 'Mailing Address' can have all the traditional information of postal addresses in the European Union.

Remark: The notion of 'Address' can be specified in many different ways[103], especially since each country has its own postal information structure. Hence, this "Address" axiomatization module is considered an "unsteady" module, and should be replaced by a more sophisticated module – one that does, for example, consider the compatibility with online national, European, or international address servers[104].

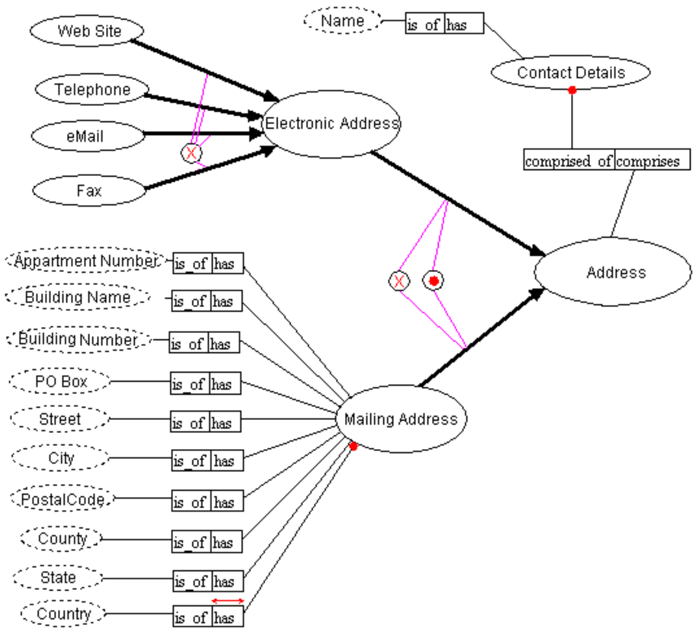


Fig. 7.7. The "Address" application axiomatization module.

[102] The concept 'Address' is defined in the CCglossary as: "A construct describing the means by which contact may be made with, or messages or physical objects may be delivered to a legal entity. An address may contain indicators for a physical or virtual (i.e. accessed electronically) location or both".
[103] Due to epistemological differences.
[104] Such address servers are: http://www.afd.co.uk/tryit/ (February 2004), http://www.postdirekt.de (February 2004), http://www.usps.com, (February 2004), etc.

7.3 Discussion and lessons learnt

This section provides a further discussion on the application of our methodological principles and tool for the development and engineering of the CContology.

Extensibility is one of the main requirements (and one of the most challenging issues) for the development of any CC-form. As we have mentioned earlier, the main goal of the CCFORM is to reach consensus about the foundation of a trusted customer complaint portal. Once such a portal is implemented as a centralized CC-form between customers and companies, companies may wish to extend "their" CC-form to inquire about more specific complaint details, e.g. delivery conditions, certain product attributes, or they might wish to offer the customer a particular resolution, etc[105]. Such extensions may be a necessity not only for individual companies but also in so called vertical markets applications (covered in the “vertical market” topic panel, TP7). In the CCFORM project, the intention is to allow companies to extend the CC-form content themselves, within given (e.g. legal) constraints on those extensions. On the one hand, this will help to achieve a wider adoption of complaint mechanisms in e-commerce applications. On the other hand, this will create new challenges such as keeping the new extensions consistent with the existing CC-form and preventing the misuse of the CC-form. For example, a company might try to misuse the CC-form by inquiring private information that violates the privacy regulations, or it may introduce new terminology and rules that are semantically inconsistent with the existing content terminology and rules.

As a solution, we propose that the CC-form not be altered directly. Instead, extensions should be introduced first into the CContology and the

[105] One can imagine a company providing a link to the CC-form portal. When the link is clicked, the CC-form appears with the company’s information filled and the details of the complaints (that are specific to this company) attached to the basic complaint questions.

base of CC-form. Moreover, our modularization of the application axiomatization part of the CContology offers simplified methodologies for extending, maintaining, and managing the CC-form:

- *Extensions will not be allowed on all axiomatization modules*. For example, the “Complainant” and “Address” axiomatization modules may be “locked”, so companies will be prevented from for example, asking privacy-rule-violating questions. Or perhaps, we can only allow extensions to be made into the “Complaint Problems” and “Complaint Resolutions” modules. In this way, we can achieve a “relatively” systematic management of the kinds of extensions allowed.

- *Extensions can be made and treated as separate modules*. If a company wishes to extend one of the seven core modules to inquire details about, for example, a certain kind of product, a new module can be constructed to capture these details. Both the core module(s) and the new module can be composed automatically. Notice that the specification of our composition operator (see section 4.4.2) guarantees that the constraints and the complaining details introduced in a core module will never be dismissed or weakened. In other words, the constraints and complaint details in the resultant composition will always imply the constraints and the complaint details in the core module.

  This is in fact a nice illustrative application of our composition mechanism, especially in the legal domain. From a “legal” viewpoint, our composition operator means that when including a module into another module (that has a higher authority, or also called *legal weight*), all rules and fact-types in the included module will be inherited by (or applied in) the including module.

- *Efficient maintenance and management*. A CC-form platform may need to manage a large number of extensions that target many different complaining issues. Releasing and treating these extensions as separate modules will make managing, maintaining and indexing them more scalable.

- *The development of the modules can be distributed among ontology experts, domain experts and application-oriented experts*. In the case of a vertical market application where one wishes to develop a set of extensions (i.e. modules), the development and the review processes can be distributed according to the expertise of the developers and the subject of the modules.

  In the development of the seven core modules we have distributed the development and review between several specialized topic panels in accordance with their expertise. Bistra Vassilev acted as domain expert for the development of the complaint problem and resolutions modules, even though she was based several thousand kilometers away. Members from TP1 (legal affairs) have contributed to the development and review of the “Complaint”, “Complainant”, “Complaint Recipient”, “Address” and “Contract” modules. Members from TP2 (consumer affairs) have similarly contributed to the development and review of the “Complaint”, “Complainant”, “Complaint Problem” and “Complaint Resolution” modules, etc.

- *Module Reusability*. Modularizing the application axiomatization of the CContology indeed simplifies the reusability of this axiomatization. One may wish to reuse some of these axiomatization modules in application scenarios other than the CC-form. For example, the ‘Address’ module can easily be reused for tasks in other domains such as Mailing, Marketing and Sales Force Automation. The `Complaint Problems’ module is in the domains of market analysis, qualitative statistics, etc.

-D

7.4 Multilingual lexicalization of the CContology

As our role in the CCFORM (through Topic Panel 6) was also to undertake the multilingual and cultural demands of customer complaint forms, a methodology for multilingual lexicalization of ontologies had to be developed. This methodology has been applied to lexicalize the CContology into several natural languages in order to support the development of a software platform providing cross-language CC-forms. For complaint platforms, this helps to systematize the translation of all terms in the generated and filled-in CC-forms that do not contain "free" text.

As shall be clear later in this section, we distinguish between a *multilingual ontology* and *multilingual lexicalization of an ontology*. The former refers either: 1) to different monolingual ontologies with an alignment layer to map between them. Such an alignment layer may include different kinds of relationships (e.g. 'equivalence', 'subtype-of', 'part-of', etc.) between concepts across the aligned ontologies. All of these ontologies, in addition to the alignment layer, form a multilingual ontology. A multilingual ontology can also be 2) a one ontology in which the terminology (i.e. concept labels) is a mixture of terms from different languages. For example, some concepts are lexicalized in language L_1, and others are lexicalized in language L_2, or maybe even in both L_1 and L_2. Yet other concepts may not have terms to lexicalize them. See [KTT03] for a methodology (called "termontography") that supports such a process of multilingual ontology engineering[106].

Multilingual lexicalization of an ontology is our aim *in this section*. It is an ontology lexicalized in a certain language (we call this the "native language") and a list of one-to-one translations of the ontology terms into

[106] The processes of modeling, engineering, or using multilingual ontologies are still open (and difficult) research issues. Some related works can be found in [LWP+02][A97a][V98][B01].

other languages. *This list is not seen as part of the ontology itself*; rather, it belongs at the application level or to a group of users.

Our approach to the multilingual lexicalization of ontologies is motivated by Avicenna's argument on the strong relationship/dependency between concepts and linguistic terms[107], and by the belief [G98a] that *an ontology is language-dependent*. Indeed, conceptual equivalence[108] between terms in different languages is very difficult to find at the domain level. Hence, from an engineering viewpoint, multilingual lexicalization (i.e. one-to-one translation) of ontology terms should not be preserved or generalized at the domain axiomatization level. Instead, such translations can be fairly established at the application level for a certain application (e.g. CC-form) or group of users.

The main goal of providing the multilingual lexicalization of an ontology is to *maximize the usability of this ontology for several cross-language applications*. We believe that this is of ever increasing importance in today's global, networked economy.

In the following paragraphs, we describe our approach to the multilingual lexicalization of ontologies using the CContology as an illustrative example.

Our approach requires *an ontology to be built and lexicalized completely in one language*, namely, the ontology's *native language*. In the case of the CContology, English is chosen as the native language that then acts as *the* reference for translating ontology terms into other languages.

Given the CCglossary (all the terms in the CContology and their glosses), and given the CC-form as a certain application scenario[109], the

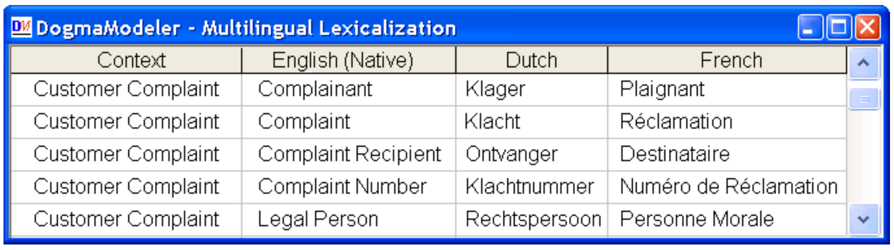
[107] See our discussion on this issue in section 3.2

[108] *Conceptual equivalence* between terms in two different languages, means that the two terms refer exactly to the same concept. This must be the case in all possible applications and/or situations where the terms appear.

[109] Notice that changing this application scenario may yield different translations.

-D

CContology has been lexicalized into 11 European languages[110]. In fig. 7.8, we provide a sample of these translations, illustrating one-to-one translation between terms in English, Dutch, and French languages.


DogmaModeler - Multilingual Lexicalization

Context	English (Native)	Dutch	French
Customer Complaint	Complainant	Klager	Plaignant
Customer Complaint	Complaint	Klacht	Réclamation
Customer Complaint	Complaint Recipient	Ontvanger	Destinataire
Customer Complaint	Complaint Number	Klachtnummer	Numéro de Réclamation
Customer Complaint	Legal Person	Rechtspersoon	Personne Morale

Fig. 7.8. An example of multilingual lexicalization of the CContology.

A CC-form can easily switch between different natural languages by substituting the terms and using the corresponding terms in such a translation list.

It is important to note that the CCglossary has played a critical role during the translation process of the CContology. The CCglossary has been used as the principal reference, by the translators[111], for understanding the intended meaning of the terms, and thus achieving better quality translations.

While it is a scalable, pragmatic, easy to use, and systemized approach, one-to-one translations are not as simple as they appear – they do sometimes yield imperfect translations. The translator needs to perform further searches in order to acquire more elegant translations. In the section that follows, we present some issues and guidelines for greater convenience and accuracy in the multilingual lexicalization of ontologies:

- *Cultural issues*. There is a great interdependency between the language and culture (social activities, religion, region, weather,

[110] These translations are not provided in this thesis as the distribution of the knowledge is restricted, and its intellectual property is owned by the CCFORM project.
[111] It is maybe worth mentioning that the translation process has been subcontracted to an a translation company whose personnel have been trained to follow our approach.

interests, etc.) of a people. Thus, within a community of people speaking the same language, we can find different usage of terms, even within the same context and situation. For example, within the "Customer Complaint" and CC-form application scenario, when translating the term "Complaint" into Arabic, there are two possible terms: "Mathalem" and "Shakaoa". In Palestine, the most commonly used term is "Shakaoa", while in Saudi Arabia, people prefer the term "Mathalem". Seemingly, the ideal solution for such a problem is to provide a set of rules for the usage of each term, considering *all* cultural issues [C98]. However, this does not yield a scalable approach for our purposes. Thus, we advise that if such cultural variations are important for a certain application scenario, it is better to treat each variation as a distinct language e.g. English-UK, English-USA, Dutch-Belgium, Dutch-Netherlands, Old-Arabic, Modern-Arabic.

- *Word to word translation is not our goal*. Usually, the purpose of building an ontology is to formally represent an agreed conceptualization of a certain domain, and share its among a community of users. Thus, lexicalizing the concepts in an ontology into multiple languages is a way of maximizing the *usability* of this ontology. It does not result in a multilingual lexicon. In lexicons or dictionaries, the purpose is to list only the common *words* (e.g. based on the corpus of a language) with a description and some lexical information. In ontologies, it is normal to find a concept lexicalized by an expression. For example, "Total Amount Paid", "Trying to obtain data improperly", etc. Such concepts cannot, in general, be lexicalized into one word - at least not in English.

To conclude, with the methodology we have presented in this chapter, we aim to maximize the usability of an ontology over several cross-language applications. This methodology is useful and easily applicable in information systems that comprise forms, database schemes, XML and RDF tags, etc. However, our methodology is not suited for ontology-based information retrieval and natural language processing applications. For such application scenarios, multilingual ontologies might be more suitable. See [GGV97][BCFF04].

7.5 Conclusions

In this chapter we have presented our experiences and main achievements in the Ontology, Extensibility multilingualism topic panel, a special interest group in the EU Thematic Network project, CCFORM.

Using ontologies as a foundation for cross-border online complaint management platforms can greatly improve the effectiveness, scope and extensibility of such platforms. While offering individual companies, organizations or associations the possibility of advanced customization (by including ontology extension capabilities) semantic consistency is maintained through the complaint management terminology. Furthermore, by restricting extensions to certain parts of the ontology, some legal constraints such as privacy regulations may be enforced systematically.

The proposed methodology for the multilingual lexicalization of ontologies is a pragmatic one. It offers a scalable way of offering multilingual services – a necessity for cross-border complaint management within the EU. An important goal in our future research is to develop a formal approach for developing multilingual ontologies which would for example, allow computers to interpret and disambiguate terms in different languages.

Chapter 8

Conclusions and Future Work

The term 'Conclusion' has 9 meanings in WordNet:

"[1] The act of ending something. [2] The act of making up your mind about something. [3] A position or opinion or judgment reached after consideration. [4] The proposition arrived at by logical reasoning (such as the proposition that must follow from the major and minor premises of a syllogism). ... "

(WordNet 1.7.1)

This final chapter concludes the thesis. We provide some discussion and concluding remarks in section 8.1 and suggest a list of related topics for future research in section 8.2.

8.1 Summary

In this thesis we have specified three foundational challenges in ontology engineering, (*viz.* ontology reusability, ontology application-independence, and ontology evolution). Based on these challenges, we have derived six engineering requirements (see section 3.5). To fulfill these requirements we have proposed two methodological principles for ontology engineering *viz.* ontology double articulation and ontology modularization. We have presented, the ORM-ML, the DogmaModeler tool prototype, and the CCFORM case study to illustrate the implementation of our methodological principles[112].

The first methodological principle suggests that an ontology be built as a domain axiomatization and its application axiomatizations. While a domain axiomatization focuses on the characterization of the intended meaning (i.e. intended models) of a vocabulary at the domain level, application axiomatizations mainly focus on the usability of this vocabulary according to certain application/usability perspectives. An application axiomatization is intended to specify the legal models - a subset of the intended model - of an application's interest.

The second methodological principle suggests that application axiomatizations be built and used in a modular manner. Axiomatizations should be developed as a set of small modules and later composed to form, and be used as, one modular axiomatization. Module composition can be performed automatically through a composition operator to combine (and imply) all axioms introduced in the composed modules.

[112] A prioritized summary of our main contributions to ontology engineering has been presented in section 1.2.

8.2 Discussion and concluding remarks

In the following tables, we present each of the six ontology-engineering requirements and summarize our methodological and implementation fulfillments.

R1	**Ontologies should be engineered in a way that allows the isolation and identification of the reusable parts of the ontology.** Fulfilling this requirement contributes to resolving the ontology reusability challenge (see section 2. 1)
Methodological fulfillment: 1. The modularization principle enables application axiomatizations to be developed and used as a set of compose-able modules, which are easier to reuse for other types of applications and tasks. (See chapter 4) 2. The double articulation principle isolates the most reusable part of an ontology (i.e. domain axiomatization) from the (more specific) application axiomatizations. (See chapter 3)	
Implementation and illustration: - The implementation of the composition operator for automating module composition, simplifies and encourages module reusability. (See section 6.6) - Two scenarios for representing modular axiomatizations have been developed. (See section 6.6) - The metadata that we have proposed is the key infrastructure for building axiomatization libraries, which enable the search, browse, management, and reuse of modules. (See our implementation of an	

axiomatization library in section 6.5)

- The CCFORM case study illustrates the development of the CContology in a modular manner. (See chapter 7). Application axiomatizations in CCFORM consists of seven modules, called core modules, extensions can be made and treated as separate modules. If a company wishes to extend one of the seven core modules to inquire details about, for example, a certain kind of product, a new module can be constructed to capture these details. Both the core module(s) and the new module can be composed automatically (i.e. reuse of the core modules). Notice that the specification of our composition operator (see section 4.4.2) guarantees that the constraints and the complaining details introduced in a core module will never be dismissed or weakened. In other words, the constraints and complaint details in the resultant composition will always imply the constraints and the complaint details in the core module. This is in fact a nice illustrative application of our composition mechanism, especially in such a legal application. From a "legal" viewpoint, our composition operator means that when including a module into another module (that has a higher authority, or also called *legal weight*), all rules and fact-types in the included module will be inherited by (or applied in) the including module.

-D

R2	**The influence of usability perspectives on ontology axioms should be well articulated, in pursuit of both reusability and usability.** Fulfilling this requirement contributes to the resolution of the ontology application-independence challenge (see section 2. 2)

Methodological fulfillment:

The double articulation principle increases the reusability of domain axiomatizations and the usability of application axiomatizations. Usability perspectives have a neglectable influence on the independency of a domain axiomatization, because ontology builders are prevented from encoding their application-specific axioms. In other words, domain axiomatizations are mainly concerned with the characterization of the "intended models" of concepts, while application axiomatizations are mainly concerned with the specification of the legal models -for a certain use- of these concepts. (See chapter 3)

Implementation and illustration:

- The DogmaModeler illustrates an intuitive approach for double-articulating axiomatizations. It shows how domain axiomatizations can be captured in the ontology base and later used to develop application axiomatizations, i.e. mapping lexons into ORM fact-types (see section 6. 2 and section 6.3).
- We have also shown how OWL can be used for representing application axiomatizations. (see section 3.4.1)
- The CCFORM case study illustrates a real-life axiomatization double-articulated as domain and application axiomatizations (see chapter 7). The CContology is engineered as a domain axiomatization, and seven

modules of application axiomatization. The intended meaning of the terminology used in these application axiomatization modules is restricted to the terminology defined at the domain axiomatization level. The application axiomatization modules are intended to play the role of conceptual data schema(s) for CC-forms development. So that, any CC-form, including its population, should be based on (i.e. commit to) the CContology through those axiomatization modules. A CC-from can be constructed manually or generated automatically (as has been illustrated in section 6.7.1). The semantics of all elements in this CC-from (i.e. the data fields) should be defined in the CContology.

- Furthermore, modularizing the application axiomatization of the CContology indeed simplifies the reusability of this axiomatization. One may wish to reuse some of these axiomatization modules in application scenarios other than the CC-form. For example, the ‘Address’ module can easily be reused for tasks in other domains such as Mailing, Marketing and Sales Force Automation. The `Complaint Problems’ module is in the domains of market analysis, qualitative statistics, etc.
- Note on the CCcontext: we have learned during the definition process of the CCcontext that it is not an easy task, and it cannot be defined rigidly in the early phases of the development of the CContology. As none of our team was an ontology expert, we provided a draft definition and investigated by providing many different examples of application scenarios that this context should cover. For example, we have questioned whether the context should cover applications such as customer-relationship-management, market analyses, sales force automation and so forth; whether it should cover all consumer regulations in any country or only in Europe; whether it should cover all commercial activity, in any place and at any time; which

-D

documents, laws and regulations should be our main references, etc. Such questions led not only to the CCcontext definition (which was achieved after several iterations), but also propelled the team to discuss deeply and even redefine the scope of the CCFORM goals.

R3	**Critical assumptions that make clear the factual meaning of an ontology vocabulary should be rendered as part of the ontology, even if informally, to facilitate both users' and developers' commonsense perception of the subject matter.** Fulfilling this requirement contributes to resolving the ontology evolution challenge (see section 2. 3)
Methodological fulfillment: 1. The notion of gloss as an auxiliary informal account of the intended meaning of a linguistic term is introduced *as part of an ontology*. It is intended to render clearly the critical assumptions, especially those that are implausible, unreasonable, or very difficult to formalize and articulate explicitly. See the definition, examples, and guidelines on how to develop a gloss in section 3.3.6. 2. The importance of using linguistic terms in investigating and rooting domain concepts is discussed and clarified. The reuse of existing lexical resources in gloss modeling is emphasized. (See section 3.2.2 and section 6.2.2).	
Implementation and illustration: - The DogmaModeler illustrates the incorporation of existing lexical resources in gloss modeling. (See section 6.2.2). - The CCFORM case study illustrates the development of the CCglossary as part of the CContology (see appendix C1). The CCglossary indeed shows how critical assumptions about a concept can be rendered informally as part of a CContology. For example, compare the gloss of (e.g. 'Legal Person', etc.) with its formal definition within the lexons. Our experience is reported in chapter 7. It is probably worth noting that intensive discussions were carried out	

-D

(by legal experts, market experts, application-oriented experts) for almost every gloss. We have found that the gloss modeling process is a great mechanism for brainstorming, domain analyses, domain understanding and for reaching (and documenting) consensus. Furthermore, it allowed non-ontology experts to participate actively in the ontology modeling process. Some partners have even noted that the CCglossary is the most useful component in the CContology. The CCglossary, which has been developed in English, has played the role of *the key* reference for lexicalizing the CContology into 11 other European languages. Translators have acknowledged that it guided their understanding of the intended meanings of the terms and allowed them to achieve better translation quality.

R4	**The ontology representation model should be capable of distributed and collaborative development.** Fulfilling this requirement contributes to resolving the ontology evolution challenge (see section 2. 3)
Methodological fulfillment: 1. The double articulation principle allows different communities to create and maintain domain and application axiomatizations. Indeed, domain experts, lexicographers, knowledge engineers, and even philosophers may contribute to the development, maintenance, and review phases of domain axiomatizations, without knowing why and how these axiomatizations will be used. Application-oriented experts may contribute to and focus on the development phases of application axiomatizations, without having any knowledge about the ontological correctness of domain axioms. 2. The modularization principle enables the distributed development of modules over different locations, expertise, and stakeholders.	
Implementation and illustration: - The DogmaModeler and ORM-ML illustrate how domain and application axiomatizations can be captured and represented in a modular and distributable manner (see chapter 5 and 6). - Our real-life experience in the distribution and collaborative development of the CContology is reported in chapter 7. *The development of the CContology modules have been distributed among ontology experts, domain experts and application-oriented experts*. In the case of a vertical market application where one wishes to develop a set of extensions (i.e. modules), the development and the review processes are distributed according to the expertise of the developers	

and the subject of the modules. In the development of the seven core modules we have distributed the development and review between several specialized topic panels in accordance with their expertise. Bistra Vassilev acted as domain expert for the development of the complaint problem and resolutions modules, even though she was based several thousand kilometers away. Members from TP1 (legal affairs) have contributed to the development and review of the "Complaint", "Complainant", "Complaint Recipient", "Address" and "Contract" modules. Members from TP2 (consumer affairs) have similarly contributed to the development and review of the "Complaint", "Complainant", "Complaint Problem" and "Complaint Resolution" modules, etc.

-D

R5 & **R6**	**Ontologies should be engineered in a way that enables smooth and efficient evolution.** **Ontologies should be engineered in a way that allows easy replacement of the axiomatization of ontology parts.** Fulfilling these two requirements contribute to resolving the ontology evolution challenge (see section 2. 3)

Methodological fulfillment:

1. The modularization principle enables application axiomatizations to evolve as modules which are easier to build, maintain, and replace. This is because the internal couplings (e.g. the number of relationships between concepts) in small modules are fewer than the internal couplings in large axiomatizations. The development and maintenance of small modules allows ontology builders a better focus and easier understanding than large and multi-domain axiomatizations. The modularity of an axiomatization also enables ontology users and maintainers to interchange some parts with others that are for example, more relevant, reliable and accurate. In short, the modularization principle indeed enables the evolution life cycle of axiomatizations to be more efficient.
2. The double articulation principle enables domain axiomatizations to grow (i.e. add lexons and glosses) without influencing application axiomatizations. (See section 6.2)
3. Glosses are a great mechanism for understanding concepts *individually*, without having to browse, reason, and understand them within an axiomatized theory. Further, compared with formal definitions, glosses help to build a “deeper” intuition about concepts by denoting implicit or tacit assumptions. This indeed makes the

evolution and maintenance of the ontology easier, especially when the ontology is particularly large-scaled, has different maintainers, or is developed over different periods (See section 3.3.6).

Implementation and illustration:

- DogmaModeler illustrate how axiomatization modules can be (de/)composed (see chapter 6).
- Our discussion and experience in the CCFORM case study illustrates the extensibility (i.e. smooth evolution) of our approach. A CC-form platform may need to manage a large number of extensions that target many different complaining issues. Releasing and treating these extensions as separate modules will make managing, maintaining and indexing them more scalable. See our discussion and lessons learnt in chapter 7.
- The unsteadiness of the "Address" axiomatization and the aim of replacing this module with other alternatives is discussed in section 7.2.

In short, our methodological principles guide ontology builders by enabling their product, i.e. ontologies, to be highly reusable and usable and easier to both build and maintain.

Contribution to ORM

Although it was not a goal of the thesis to contribute to conceptual data modeling approaches, we have encountered several possible improvements and extensions to ORM which might be used outside the ontology engineering context. These include: composition of ORM schemes; including constraint patterns for reasoning about the satisfiability of ORM schemes; developing ORM-ML for representing ORM schemes in a textual manner; developing verbalization templates for verbalizing ORM schemes into English, Dutch, Arabic, and Russian; the mapping of ORM schemes into web forms; enabling ORM to be reused for other purposes than database modeling, *viz.* ontology engineering.

In the same way, we believe that other conceptual data modeling approaches (such as EER and UML) can benefit from theses developments.

8.3 Future Research

In relation to the subject matter of this thesis, the following are suggested as worthy future research topics:

1. *Incorporate primitives of Upper Level Ontologies in domain axiomatizations*. As we have shown in section 3.3.7, the formalization of lexons might be not enough for achieving *systematic ontological quality* on the specification of the intended meanings of linguistic terms. These specifications might need to receive further formal restrictions. For this, in section 3.3.7 we have proposed to incorporate upper level ontologies at the domain axiomatization level. We have introduced the notions of "Term upper-forms" and "Lexon upper-forms", so that the formal definitions of superior types of concepts and relationships (that can be found in upper level ontologies) can be induced into Terms and Lexons, respectively. In an upcoming effort, we plan to extend our DogmaModeler tool by developing a library of upper-ontology components (especially for DOLCE), so that ontology builders will be able to plug-in and automatically reason about the quality of their lexons. See section 6.2.2, and section 6.2.3.
2. *Investigate how to validate and deal with the lexical issues of ontology terms*. For example, in the following lexon <**Bibliography: Book, issuedBy, Issues, Publish**>, one can spot, lexically, that the term "**Publish**" is improper as it is a "verb". Some lessons on how to validate and deal with the lexical issues of the ontology vocabulary can be learned from the "lexical semantics" research community[113], such as, the use of nouns and adjectives verses terms, verbs verses roles, the modeling of idioms, the specific uses of metaphors, singulars, plurals, classification of ontology roles

[113] Specially from the emerging WordNet-alike (or so called "mental lexicons") communities, such as http://www.globalwordnet.org/ (January, 2005).

verses classification of verbs, term stemming, spell checking, etc. See section 6.4.

3. *Include more lexical resources into DogmaModeler (or its DogmaStudio[114] successor) to support the gloss modeling process*. As we have discussed in section 3.5, many existing lexical resources (such as lexicons, glossaries, thesaurus, dictionaries, etc.) are indeed important sources of glosses. For adaptation and reusability of such resources: 1) we would plan to implement a full adaptation of WordNet-alike lexicons into DogmaModeler. See section 6.2 for the current support and illustration of this functionality. In addition, as gloss has a strict intention in our approach and so that not every lexical resource can be adopted (i.e. it should provide a clear discrimination of word/term meaning(s) in a machine-referable manner), 2) we plan to investigate how other kinds of lexicons and dictionaries such as the Cambridge dictionary can be ontologized and adopted: extract and re-engineer their meaning descriptions into machine-referable glosses, and so excluding the typical morphological and lexical issues. See section 3.5 and section 6.2.

4. *Develop a methodology for developing multilingual ontologies*. The methodology that we have presented in section 7.4 is aimed with the maximization of the usability of an ontology over cross-language applications. This methodology is useful and easily applicable in information systems that comprise forms, database schemes, XML and RDF tags, etc. However, this methodology is not suited for other application scenarios such as ontology-based information retrieval, natural language processing, etc. For such application scenarios, multilingual ontologies might be more suitable. A multilingual ontology is an ontology in which the

[114] DogmaStudio is an initiative to re-implement DogmaModeler using the Eclipse environment.

terminology (i.e. concept labels) is a mixture of terms from different languages. In the future, we plan to develop a methodology for building such multilingual ontologies, and we plan to extend DogmaModeler for this regard. See section 7.4.

5. Develop a step-wise methodology for ontology development. The ontology engineering approach that have been presented in this thesis is not yet equipped with a step-wise methodology. Such a methodology is supposed to provide guide for ontology builders by dividing the ontology development process (of both domain and application axiomatizations) into a set of phases and a series of steps and guidelines to be followed in each phase. This methodology should take into account 1) the simplicity of the ontology modeling process, 2) the quality of the ontology content being modeled (perusing both usability and reusability), 3) the distribution of ontology evolution, etc. Some lessons can learnt from the AKEM Methodology [ZKK+04] or other existing methodologies such as Methontology, On-To-Knowledge [S03b], Methontology [FGJ97], etc. See section 1.2.

6. Include other languages in the DogmaModeler or its successor for representing application axiomatizations. At this stage, DogmaModeler supports the modeling of application axiomatizations using only ORM as a specification langauge. To increase the usability of application axiomatizations, DogmaModeler should allow these axiomatizations to be specified in multiple specification languages, such as DAML+OIL, OWL, RuleML, EER, UML, Ω-RIDL, etc. Indeed, ORM is mainly suitable for database and XML (-based) application scenarios since it is quite comprehensive in its treatment of the integrity of data sets. For inference and reasoning application scenarios, description logic based languages (such as OWL, DAML, etc.) seem to be more applicable than other languages, as they focus on the

-D

expressiveness and the decidability of axioms. See section 3.4.1. As an upcoming activity, we plan to extend DogmaModeler to support, at least OWL-Lite, and import-export functionalities into several languages.

7. *Map ORM into the* ***DLR*** *Description Logic*. In this way, the satisfiability of ORM schemes can be completely verified. As we have noted earlier in section 4.5, the general problem of determining the consistency for all possible constraint patterns in ORM is un-decidable [H97], and hence neither our ORM composition algorithm nor our logical validations in DogmaModeler can be complete. Therefore, a complete semantic tableaux algorithm for deciding the satisfiability of ORM schemes is needed. To achieve this we plan to reformalize ORM by mapping all of its primitives and constraints into the ***DLR*** Description Logic [CDLNR98]. ***DLR*** is a powerful and decidable fragment of first order logic. It supports general inclusion axioms, inverse roles, number-restrictions, reflexive-transitive closure of roles, fixpoint constructs for recursive definitions, relations of arbitrary arity, etc.

Appendices

Appendix A: ORM Markup Language

This appendix presents the XML-Schema for the ORM Markup Language, as the grammar reference of ORM-ML documents. This schema is an intensively improved version (Ver.2) of the ORM-ML XML-schema that we have published earlier in [DJM02a][DJM02b] and [JDM03]. In appendix A1 we present a tree view of the ORM-ML XML-schema, and in appendix A2 we present the ORM-ML XML-schema. Appendix A3 presents a complete example, as an instance of this schema.

Appendix A1 (tree view of the ORM-ML XML-Schema)

A tree view of the elements in the XML Schema is given in Appendix A2. Please note the attributes of the elements are omitted here for clarity of presentation.

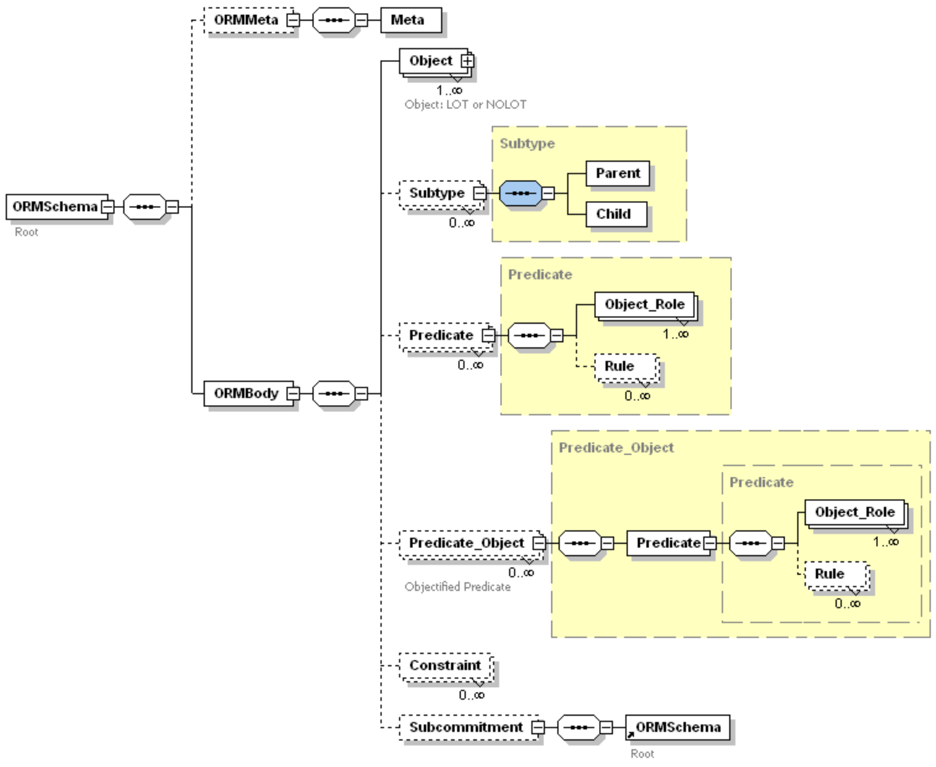


Fig. A.1. A tree view of the elements in the ORM-ML XML Schema.

Appendix A2 (ORM-ML XML-Schema)

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <!--  edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by rth77 (rth77)
  -->
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dc="http://purl.org/dc/elements/1.1/" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://purl.org/dc/elements/1.1/"
schemaLocation="http://www.ukoln.ac.uk/metadata/dcmi/dcxml/xmls/dc.xsd" />
- <xs:element name="ORMSchema">
- <xs:annotation>
  <xs:documentation>Root</xs:documentation>
  </xs:annotation>
- <xs:complexType>
- <xs:complexContent>
- <xs:extension base="ORMType">
- <xs:sequence>
- <xs:element name="ORMMeta" minOccurs="0">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="Meta">
- <xs:complexType>
  <xs:attribute name="Name" type="xs:string" use="required" />
  <xs:attribute name="Content" type="xs:string" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="ORMBody">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="Object" type="Object" maxOccurs="unbounded">
- <xs:annotation>
  <xs:documentation>Object: LOT or NOLOT</xs:documentation>
  </xs:annotation>
  </xs:element>
  <xs:element name="Subtype" type="Subtype" minOccurs="0" maxOccurs="unbounded"
/>
  <xs:element name="Predicate" type="Predicate" minOccurs="0"
maxOccurs="unbounded" />
- <xs:element name="Predicate_Object" type="Predicate_Object" minOccurs="0"
maxOccurs="unbounded">
- <xs:annotation>
  <xs:documentation>Objectified Predicate</xs:documentation>
  </xs:annotation>
  </xs:element>
  <xs:element name="Constraint" type="Constraint" minOccurs="0"
maxOccurs="unbounded" />
- <xs:element name="Subcommitment" minOccurs="0">
- <xs:complexType>
- <xs:sequence>
  <xs:element ref="ORMSchema" />
```

-D

```
  </xs:sequence>
  <xs:attribute name="order" type="xs:integer" use="optional" />
  <xs:attribute name="URI" type="xs:string" use="optional" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="OntologyBase" type="xs:string" use="required" />
  <xs:attribute name="Context" type="xs:string" use="required" />
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
  </xs:element>
- <xs:complexType name="Object" abstract="true">
- <xs:annotation>
  <xs:documentation>Object: LOT or NOLOT</xs:documentation>
  </xs:annotation>
- <xs:sequence>
- <xs:element name="Translation" minOccurs="0" maxOccurs="unbounded">
- <xs:complexType>
  <xs:attribute name="Language" type="xs:string" use="required" />
  <xs:attribute name="Name" type="xs:string" use="required" />
  <xs:attribute name="Description" type="xs:string" use="required" />
  <xs:attribute name="Reference" type="xs:string" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="Name" type="xs:ID" use="required" />
  <xs:attribute name="Gloss" type="xs:string" use="optional" />
  <xs:attribute name="Datatype" type="xs:string" use="optional" />
  <xs:attribute name="TermUpperForm" type="xs:string" use="optional" />
  <xs:attribute name="NameSpace" type="xs:string" use="optional" />
  </xs:complexType>
- <xs:complexType name="LOT">
- <xs:annotation>
  <xs:documentation>Lexical Object Type</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Object">
  <xs:attribute name="numeric" type="xs:boolean" use="optional" default="false" />
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="NOLOT">
- <xs:annotation>
  <xs:documentation>Non Lexical Object Type</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Object">
- <xs:sequence>
- <xs:element name="Reference" minOccurs="0">
- <xs:complexType>
```

```
  <xs:attribute name="Ref_Name" use="required" />
  <xs:attribute name="numeric" type="xs:boolean" use="optional" default="false" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="Independent" type="xs:boolean" use="optional" default="false" />
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Object_Role">
- <xs:annotation>
  <xs:documentation>Object + Role</xs:documentation>
  </xs:annotation>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="Object" type="xs:IDREF" use="required" />
  <xs:attribute name="Role" type="xs:string" use="optional" />
  </xs:complexType>
  <xs:complexType name="ORMType" />
- <xs:complexType name="Predicate">
- <xs:sequence>
  <xs:element name="Object_Role" type="Object_Role" maxOccurs="unbounded" />
  <xs:element name="Rule" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="Derived" type="xs:boolean" default="false" />
  <xs:attribute name="Derived_Stored" type="xs:boolean" default="false" />
  </xs:complexType>
- <xs:complexType name="Constraint" abstract="true">
- <xs:annotation>
  <xs:documentation>Abstract element for constraints</xs:documentation>
  </xs:annotation>
  </xs:complexType>
- <xs:complexType name="Predicate_Object">
- <xs:annotation>
  <xs:documentation>Objectified Predicate</xs:documentation>
  </xs:annotation>
- <xs:sequence>
  <xs:element name="Predicate" type="Predicate" />
  </xs:sequence>
  <xs:attribute name="Predicate_Name" type="xs:ID" use="required" />
  </xs:complexType>
- <xs:complexType name="Subtype">
- <xs:annotation>
  <xs:documentation>SubType</xs:documentation>
  </xs:annotation>
- <xs:sequence>
- <xs:element name="Parent">
- <xs:complexType>
  <xs:attribute name="Object" type="xs:IDREF" />
  <xs:attribute name="Role" type="xs:string" />
  </xs:complexType>
  </xs:element>
- <xs:element name="Child">
- <xs:complexType>
  <xs:attribute name="Object" type="xs:IDREF" />
  <xs:attribute name="Role" type="xs:string" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:complexType>
- <xs:complexType name="Mandatory">
- <xs:annotation>
  <xs:documentation>Mandatory Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Uniqueness">
- <xs:annotation>
  <xs:documentation>Uniqueness Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Subset">
- <xs:annotation>
  <xs:documentation>SubSet Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
- <xs:element name="Parent">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="Child">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
```

```
- <xs:complexType name="Equality">
- <xs:annotation>
  <xs:documentation>Equality Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
- <xs:element name="First">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="Second">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Exclusion">
- <xs:annotation>
  <xs:documentation>Exclusion Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
- <xs:element name="First">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="Second">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Frequency">
- <xs:annotation>
  <xs:documentation>Frequency Constraint</xs:documentation>
  </xs:annotation>
```

-D

```
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="Minimum" type="xs:integer" />
  <xs:attribute name="Maximum" type="xs:integer" />
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Irreflexive">
- <xs:annotation>
  <xs:documentation>Irreflexive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Intransitive">
- <xs:annotation>
  <xs:documentation>Intransitive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Transitive">
- <xs:annotation>
  <xs:documentation>Transitive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Acyclic">
- <xs:annotation>
  <xs:documentation>Acyclic Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Type" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Asymmetric">
- <xs:annotation>
  <xs:documentation>Assymetric Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Antisymmetric">
- <xs:annotation>
  <xs:documentation>Antisymmetric Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Symmetric">
- <xs:annotation>
  <xs:documentation>Symmetric Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Reflexive">
- <xs:annotation>
  <xs:documentation>Reflexive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
  <xs:extension base="Constraint" />
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Total">
- <xs:annotation>
  <xs:documentation>Total constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Supertype" />
  <xs:element name="Subtype" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Exclusive">
- <xs:annotation>
  <xs:documentation>Exclusive constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Supertype" />
  <xs:element name="Subtype" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Value">
- <xs:annotation>
  <xs:documentation>Exclusive constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
- <xs:element name="Value" maxOccurs="unbounded">
- <xs:complexType>
  <xs:attribute name="datatype" type="xs:string" use="required" />
  </xs:complexType>
  </xs:element>
- <xs:element name="ValueRange" maxOccurs="unbounded">
- <xs:complexType>
  <xs:attribute name="datatype" type="xs:string" use="required" />
  <xs:attribute name="begin" type="xs:string" use="required" />
  <xs:attribute name="end" type="xs:string" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Partition">
- <xs:annotation>
  <xs:documentation>Partition constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Subtype" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="Supertype" type="xs:IDREF" use="required" />
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Intransitive_symmetric">
- <xs:annotation>
  <xs:documentation>Intransitive + symmetric Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Acyclic_intransitive">
- <xs:annotation>
  <xs:documentation>Acyclic+intransitive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Asymmetric_intransitive">
- <xs:annotation>
  <xs:documentation>Asymmetric+intransitive Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="Irreflexive_symmetric">
- <xs:annotation>
  <xs:documentation>Irreflexive + symmetric Ring Constraint</xs:documentation>
  </xs:annotation>
- <xs:complexContent>
- <xs:extension base="Constraint">
- <xs:sequence>
  <xs:element name="Object_Role" type="xs:IDREF" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
  </xs:schema>
```

Appendix A3: Complete Example

A complete example of an ORM schema diagram (Appendix A3.1), with the associated ORM-ML document (Appendix A3.2), and ORM pseudo NL generated by the DogmaModeler (Appendix A3.3).

Appendix A3.1: ORM Schema diagram

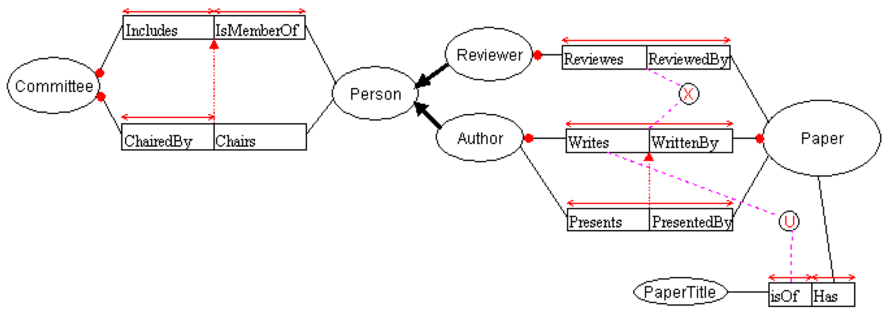


Fig. A.2. ORM schema diagram example

Appendix A3.2: Corresponding ORM-ML

```
<?xml version='1.0' encoding='UTF-8'?>
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.starlab.vub.ac.be/staff/alisovoy/ormml.xsd
' xmlns:dc='http://purl.org/dc/elements/1.1/' OntologyBase="Publishing"
Context="Scientific Conference">

<ORMMeta>
 <Meta name="DC.title" content="ORM ML example"/>
 <Meta name="DC.creator" content="Mustafa Jarrar"/>
 <Meta name="DC.description" content="A complete example of an ORM ML file"/>
</ORMMeta>
<ORMBody>
<Object xsi:type='NOLOT' Name='Committee'/>

<Object xsi:type='NOLOT' Name='Person'/>
<Object xsi:type='NOLOT' Name='Author'/>
<Object xsi:type='NOLOT' Name='Reviewer'/>
<Object xsi:type='NOLOT' Name='Paper'/>
<Object xsi:type='NOLOT' Name='PaperTitle' />
<Subtype>
 <Parent Object="Person" Role="Types"/>
 <Child Object="Author" Role="IsA"/>
</Subtype>
<Subtype>
 <Parent Object="Person" Role="Types"/>
 <Child Object="Reviewer" Role="IsA"/>
```

-D

```
</Subtype>
<Predicate>
 <Object_Role ID='ORM ML example:42' Object='Committee' Role='Includes'/>
 <Object_Role ID='ORM ML example:43' Object='Person' Role='IsMemberOf'/>
</Predicate>
<Predicate>
 <Object_Role ID='ORM ML example:44' Object='Committee' Role='ChairedBy'/>
 <Object_Role ID='ORM ML example:45' Object='Person' Role='Chairs'/>
</Predicate>
<Predicate>
 <Object_Role ID='ORM ML example:46' Object='Reviewer' Role='Reviewes'/>
 <Object_Role ID='ORM ML example:47' Object='Paper' Role='ReviewedBy'/>
</Predicate>
<Predicate>
 <Object_Role ID='ORM ML example:48' Object='Author' Role='Writes'/>
 <Object_Role ID='ORM ML example:49' Object='Paper' Role='WrittenBy'/>
</Predicate>
<Predicate>
 <Object_Role ID='ORM ML example:50' Object='Author' Role='Presents'/>
 <Object_Role ID='ORM ML example:51' Object='Paper' Role='PresentedBy'/>
</Predicate>
<Predicate>
 <Object_Role ID='ORM ML example:52' Object='PaperTitle' Role='isOf'/>
 <Object_Role ID='ORM ML example:53' Object='Paper' Role='Has'/>
</Predicate>

<Constraint xsi:type='Mandatory'>
 <Object_Role>ORM ML example:42</Object_Role>
</Constraint>
<Constraint xsi:type='Mandatory'>
 <Object_Role>ORM ML example:44</Object_Role>
</Constraint>
<Constraint xsi:type='Mandatory'>
 <Object_Role>ORM ML example:46</Object_Role>
</Constraint>
<Constraint xsi:type='Mandatory'>
 <Object_Role>ORM ML example:49</Object_Role>
</Constraint>
<Constraint xsi:type='Mandatory'>
 <Object_Role>ORM ML example:48</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:42</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:44</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:43</Object_Role>
</Constraint>
<Constraint xsi:type='Subset'>
 <Parent>
  <Object_Role>ORM ML example:42</Object_Role>
  <Object_Role>ORM ML example:43</Object_Role>
 </Parent>
 <Child>
  <Object_Role>ORM ML example:44</Object_Role>
  <Object_Role>ORM ML example:45</Object_Role>
 </Child>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:50</Object_Role>
 <Object_Role>ORM ML example:51</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:48</Object_Role>
 <Object_Role>ORM ML example:49</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:46</Object_Role>
 <Object_Role>ORM ML example:47</Object_Role>
</Constraint>
<Constraint xsi:type='Exclusion'>
 <First>
  <Object_Role>ORM ML example:48</Object_Role>
  <Object_Role>ORM ML example:49</Object_Role>
 </First>
 <Second>
  <Object_Role>ORM ML example:46</Object_Role>
  <Object_Role>ORM ML example:47</Object_Role>
 </Second>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:48</Object_Role>
 <Object_Role>ORM ML example:52</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:53</Object_Role>
</Constraint>
<Constraint xsi:type='Uniqueness'>
 <Object_Role>ORM ML example:52</Object_Role>
</Constraint>
<Constraint xsi:type='Subset'>
 <Parent>
  <Object_Role>ORM ML example:48</Object_Role>
  <Object_Role>ORM ML example:49</Object_Role>
 </Parent>
 <Child>
  <Object_Role>ORM ML example:50</Object_Role>
  <Object_Role>ORM ML example:51</Object_Role>
 </Child>
</Constraint>
</ORMBody>
</ORMSchema>
```

Appendix A3.3: Corresponding ORM Verbalization

The following are Pseudo NL sentences, generated by the DogmaModeler, as verbalizations of the ORM schema diagram.

●	Each Committee must ChairedBy at least one Person.
●	Each Committee must Includes at least one Person.
●	Each Reviewer must Reviewes at least one Paper.
●	Each Author must Writes at least one Paper.
●	Each Paper must WrittenBy at least one Author.
↔	Each Paper must Has at most one PaperTitle.
↔	Each PaperTitle must isOf at most one Paper.
↔	Each Committee must ChairedBy at most one Person.
⟵	It is disallowed that the same Committee Includes the same Person more then once, and it is disallowed that the same Person IsMemberOf the same Committee more then once.
⟵	It is disallowed that the same Author Presents the same Paper more then once, and it is disallowed that the same Paper PresentedBy the same Author more then once.
⟵	It is disallowed that the same Author Writes the same Paper more then once, and it is disallowed that the same Paper WrittenBy the same Author more then once.
⟵	It is disallowed that the same Reviewer Reviewes the same Paper more then once, and it is disallowed that the same Paper ReviewedBy the same Reviewer more then once.
↑	Each Person who Chairs a Committee must also IsMemberOf that Committee.
↑	Each Paper who WrittenBy a Author must also PresentedBy that Author.
⊗	Each Paper which is WrittenBy a Person must not ReviewedBy with that Person.
Ⓤ	Each (PaperTitle, Author) as a combination refers to at most one Paper.

Appendix B: DogmaModeler

Appendix B1: DogmaModeler Ontology Metadata

In this appendix we present the glossary of the DogmaModeler Metadata elements.

Element Name	Gloss
Acronym	An abbreviation formed from the initial letter or letters of words in the ontology title. E.g. 'CCOntology', or 'DOLCE'.
Title	The full and official heading or name of the ontology. It gives a brief summary of the matters it deals with. E.g. 'Customer Complaint Ontology', or 'Descriptive Ontology for Linguistic and Cognitive Engineering'.
Version	Information about the edition of this ontology. Typically, it includes the version number, label, and date. Whenever the ontology is enhanced, updated or improved, it is often assigned a new version. Although versions represent the different states of an ontology during its life cycle, different versions are seen as different ontologies.
Number	A unique code assigned to the ontology for identification. This number is usually assigned by an ontology registration entity.
URI	Uniform Resource Identifier, the W3C's

-D

	codification of the address syntax of an ontology. In its most basic form, a URI consists of a scheme name (such as file, http, ftp) followed by a colon, followed by a path whose nature is determined by the scheme that precedes it (see RFC 1630). URI is the umbrella term for URNs, URLs, and all other Uniform Resource Identifiers.
Genericity	The level of generalization of an the ontology. The genericity level of an ontology is typically one of the {‘Application’, ‘task’, ‘Domain’, ‘Core’, ‘Foundational’, ‘Linguistic’, ‘Metamodel’}. Examples: The CCOntology is a ‘core’ ontology; DOLCE is a ‘foundational’ ontology; “WordNet” is a ‘Linguistic’ Ontology. etc.
Language	The human language in which the ontology terms (i.e. labels of concepts, roles, etc) is expressed. In case this terminology is expressed in more than one language, the value of this attribute is ‘Multilingual’. The best practice recommended is the use of RFC 3066 [RFC3066] which, in conjunction with ISO639 [ISO639]), defines two- and three-letter primary language tags with optional subtags. Examples include "en" or "eng" for English, "akk" for Akkadian", and "en-GB" for English as it is used in the United Kingdom.
DevelopmentStatus	The completion status or condition of this

-D

	ontology, typically one of {Draft, Final, Revised, Unavailable}.
DomainSubject	A heading descriptor indicating the subject matter and the domain of the ontology. For example, e-business, sport, book-shopping and car-rental. Typically, doman subjects are expressed as keywords, key phrases, or classification codes. The recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.
Context	Information about of the scope of the ontology, in which the interpretation (i.e. the intended meaning) of the ontology terminology is bounded. For example: the context of the WordNet ontology could be the English language, the context of the "CCOntology" is the EU complaint regulations, etc.
Description	Further information about the ontology. It may include but is not limited to: an abstract, reference to a graphical representation, a free-text account of the content, the methodology used to build this ontology, documentation, etc.
Creator	An entity primarily responsible for creating the ontology. Examples of creators include persons, organizations and services. Typically, the name of a creator should be used to indicate the entity.

Contributor	An entity responsible for making contributions to the ontology content. Examples of a Contributor include a person, an organization, and a service. Typically, the name of a contributor should be used to indicate the entity.
CreationDate	The date that is associated with the creation of the ontology. In other words, the first date in the ontology lifecycle. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 [W3CDTF] and includes (among others) dates of the form YYYY-MM-DD.
Rights	Information about rights held in and over the ontology. Typically, rights will contain a copyrights statement and other restriction for the ontology, and the cost description in case the use of this ontology requires payment. If the Rights element is absent, no assumptions may be made about any rights held in or over the resource.
SpecificationLanguage	The formal language in which the ontology is being specified; for example, OWL, DAML-OIL, ORM-ML, UML, KIF, etc.
Validation	An evidence about the testing activities of the ontological content. Such tests might be conceptual or ontological quality, syntax validation, etc. Typically, one should indicate the validation methodology and comments

	about the results.
Tool	The name of the tool by which the ontology has been developed, e.g. Protégé, DogmaModeler, etc.
Application	Citation to the application(s) using/has used this ontology. Typically, one should provide the name, URL, and some description about the application.
NumberOfConcepts	Statistics about the number of concepts in the ontology.
NumberOfRelations	Statistics about the number of relations in the ontology.
NumberOfAxioms	Statistics about the number of axioms in the ontology - an axiom is typically a formal definition/expression.
NumberOfInstances	Statistics about the number instances in the ontology.
IncludesOntology/ IncludedInOntology	A reference to another ontology, which is supposed to be included as part of this ontology. Examples of such relations between ontologies include “Imports” in OWL, “inclusion” in Ontolingua and “Compose” in DogmaModeler. The formal semantics of such relationships are necessarily the same.
StepVersionOf/	A reference to the step/previous version of this

-D

PreviousVersionOf	ontology.

Appendix B2: XML-Schema of ORM-ML graphical style sheets

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dc="http://purl.org/dc/elements/1.1/" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://purl.org/dc/elements/1.1/"
schemaLocation="http://www.ukoln.ac.uk/metadata/dcmi/dcxml/xmls/dc.xsd" />
- <xs:element name="ORMGSSchema">
- <xs:annotation>
  <xs:documentation>Root</xs:documentation>
  </xs:annotation>
- <xs:complexType>
- <xs:complexContent>
- <xs:extension base="ORMType">
- <xs:sequence>
- <xs:element name="ORMMeta" minOccurs="0">
- <xs:complexType>
- <xs:sequence>
  <xs:element ref="dc:title" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:creator" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:subject" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:description" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:publisher" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:contributor" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:date" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:type" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:format" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:identifier" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:source" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:language" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:relation" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:coverage" minOccurs="0" maxOccurs="unbounded" />
  <xs:element ref="dc:rights" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="ORMBody">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="Object" type="Object" minOccurs="0" maxOccurs="unbounded" />
  <xs:element name="Predicate" type="Predicate" minOccurs="0"
maxOccurs="unbounded" />
  <xs:element name="ORConnector" type="ORConnector" minOccurs="0"
maxOccurs="unbounded" />
  <xs:element name="Subtype" type="Subtype" minOccurs="0" maxOccurs="unbounded"
/>
  <xs:element name="Subset" type="Subset" minOccurs="0" maxOccurs="unbounded" />
  <xs:element name="Text" type="Text" minOccurs="0" maxOccurs="unbounded" />
  <xs:element name="ExUniqueness" type="ExUniqueness" minOccurs="0"
```

-D

```
maxOccurs="unbounded" />
  <xs:element name="ExMandatory" type="ExMandatory" minOccurs="0"
maxOccurs="unbounded" />
  <xs:element name="Mandatory" type="Mandatory" minOccurs="0"
maxOccurs="unbounded" />
  <xs:element name="Equality" type="Equality" minOccurs="0" maxOccurs="unbounded"
/>
  <xs:element name="Exclusion" type="Exclusion" minOccurs="0"
maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
  </xs:element>
- <xs:complexType name="Object" abstract="true">
  <xs:attribute name="name" type="xs:ID" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  <xs:attribute name="width" use="required" />
  <xs:attribute name="height" use="required" />
  <xs:attribute name="ColorRGB" use="required" />
  </xs:complexType>
- <xs:complexType name="Predicate">
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  <xs:attribute name="width" use="required" />
  <xs:attribute name="height" use="required" />
  <xs:attribute name="ColorRGB" use="required" />
  </xs:complexType>
- <xs:complexType name="ORConnector">
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="1" maxOccurs="1">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
- <xs:element name="Object" minOccurs="1" maxOccurs="1">
- <xs:complexType>
  <xs:attribute name="name" type="xs:IDREF" use="required" />
  </xs:complexType>
  </xs:element>
- <xs:element name="Mandatory" minOccurs="0" maxOccurs="1">
- <xs:complexType>
  <xs:attribute name="RoleID" type="xs:IDREF" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  </xs:complexType>
```

-D

```
- <xs:complexType name="SubType">
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="ChildObjectID" type="xs:IDREF" use="required" />
  <xs:attribute name="ParentObjectID" type="xs:IDREF" use="required" />
  </xs:complexType>
- <xs:complexType name="Subset">
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="2" maxOccurs="2">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  </xs:complexType>
- <xs:complexType name="Text" abstract="true">
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="content" type="xs:string" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  <xs:attribute name="width" use="required" />
  <xs:attribute name="height" use="required" />
  </xs:complexType>
- <xs:complexType name="ExUniqueness">
- <xs:annotation>
  <xs:documentation>External Uniqueness</xs:documentation>
  </xs:annotation>
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="2" maxOccurs="2">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  </xs:complexType>
- <xs:complexType name="ExMandatory">
- <xs:annotation>
  <xs:documentation>External Mandatory</xs:documentation>
  </xs:annotation>
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="2" maxOccurs="2">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  </xs:complexType>
- <xs:complexType name="Mandatory" abstract="true">
  <xs:attribute name="RoleID" type="xs:IDREF" use="required" />
  </xs:complexType>
- <xs:complexType name="Equality">
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="2" maxOccurs="2">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  </xs:complexType>
- <xs:complexType name="Exclusion">
- <xs:sequence>
- <xs:element name="Predicate" minOccurs="2" maxOccurs="2">
- <xs:complexType>
  <xs:attribute name="ID" type="xs:IDREF" use="required" />
  <xs:attribute name="PortType" use="required" />
  </xs:complexType>
  </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required" />
  <xs:attribute name="x" use="required" />
  <xs:attribute name="y" use="required" />
  </xs:complexType>
  </xs:schema>
```

-D

Appendix B3: ORM Verbalization Templates

In this appendix, we provide 3 verbalization templates for English, Dutch, Arabic, and Russian, respectively. Each template is illustrated with an ORM diagram and its resultant constraint verbalizations, as generated by DogmaModeler.

English verbalization template

```
<?xml version='1.0' encoding='UTF-8'?>
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.jarrar.info/orm/verbalization/'>
<ORMNLMeta>
 <Meta name="DC.Title" content="English verbalization template"/>
 <Meta name="DC.Version" content="0.3"/>
 <Meta name="DC.Creator" content="Mustafa Jarrar"/>
 <Meta name="DC.Language" content="English"/>
 </ORMNLMeta>

<ORMNLBody>
<Constraint xsi:type="Lexical">
 <Text>  -Lexical concepts are :{</Text>
 <Object index="0"/>
 <Loop index="1">
  <Text>,</Text>
  <Object index="n"/>
 </Loop>
 <Text>}</Text>
</Constraint>

<FactType xsi:type="FactType">
 <Text>   -</Text>
 <Object index="0"/>
 <Role index="0"/>
 <Text>/</Text>
 <Role index="1"/>
 <Object index="1"/>
</FactType>

<Constraint xsi:type="Mandatory">
 <Text> -[Mandatory]  Each</Text>
 <Object index="0"/>
 <Text>must</Text>
 <Role index="0"/>
 <Text>at least one</Text>
 <Object index="1"/>
</Constraint>
```

```
<Constraint xsi:type="Backward Mandatory">
 <Text> -[M]  For each</Text>
 <Object index="0"/>
 <Text>there is at least one</Text>
 <Object index="1"/>
 <Text>that</Text>
 <Role index="1"/>
 <Text>this</Text>
 <Object index="0"/>
</Constraint>

<Constraint xsi:type="Disjunctive Mandatory">
 <Text> -[Mandatory]  Each</Text>
 <Object index="0"/>
 <Text>should be</Text>
 <Role index="0"/>
 <Object index="1"/>
 <Loop index="1" >
  <Text>or</Text>
  <Role index="n"/>
  <Object index="n"/>
 </Loop>
</Constraint>

<Constraint xsi:type="Uniqueness">
 <Text> -[Uniqueness]  Each</Text>
 <Object index="0"/>
 <Text>must</Text>
 <Role index="0"/>
 <Text>at most one</Text>
 <Object index="1"/>
</Constraint>

<Constraint xsi:type="Backward Uniqueness">
 <Text> -[Uniqueness]  For each</Text>
 <Object index="0"/>
 <Text>there must be at most one</Text>
 <Object index="1"/>
 <Text>that</Text>
 <Role index="1"/>
 <Text>this</Text>
 <Object index="0"/>
</Constraint>

<Constraint xsi:type="Many Uniqueness">
 <Text> -[Uniqueness]  It is possible that </Text>
 <Object index="0"/>
<Role index="0"></Role>
 <Text>more than one</Text>
 <Object index="1"/>
 <Text>, and vice versa</Text>
```

-D

```
</Constraint>

<Constraint xsi:type="External Uniqueness">
 <Text> -[Uniqueness]  The combination of {</Text>
 <Object index="1"/>
 <Loop index="1">
  <Text>and</Text>
  <Object index="n"/>
 </Loop>
 <Text>} must refer to at most one</Text>
 <Object index="0"/>
</Constraint>

<Constraint xsi:type="Subtype">
 <Text> -[Subtype]  Each instance of</Text>
 <Object index="child"/>
 <Text>is also an instance of</Text>
 <Object index="parent"/>
</Constraint>

<Constraint xsi:type="Value">
 <Text> -[Value]  The possible instances of </Text>
 <Object index="0"/>
 <Text> are :{</Text>
 <Value index="0"/>
 <Loop index="1">
  <Text>,</Text>
  <Value index="n"/>
 </Loop>
 <Text> }</Text>
 </Constraint>

<Constraint xsi:type="Exclusive">
<Text> -[Exclusive]  Each</Text>
<Object index="0"/>
<Text>should be either</Text>
<Object index="1"/>
<Loop index="1">
  <Text>or</Text>
  <Object index="n"/>
</Loop>
</Constraint>

<Constraint xsi:type="Total">
 <Text> -[Totality]  Each</Text>
 <Object index="0"/>
 <Text>must be, at least, </Text>
 <Object index="1"/>
 <Loop index="1">
  <Text>or</Text>
  <Object index="n"/>
 </Loop>
</Constraint>

<Constraint xsi:type="Partition">
 <Text> -[Partition]  Each</Text>
 <Object index="0"/>
 <Text>is at least one of</Text>
 <Object index="1"/>
 <Loop index="1">
  <Text>or</Text>
  <Object index="n"/>
 </Loop>
 <Text>but not all</Text>
</Constraint>

<Constraint xsi:type="Subset">
 <Text> -[Subset]  If</Text>
 <Object index="0"/>
 <Role index="child"/>
 <Object index="child"/>
 <Text>then this</Text>
 <Object index="0"/>
 <Role index="parent"/>
 <Object index="parent"/>
</Constraint>

<Constraint xsi:type="Subset FactType">
 <Text> -[Subset]  If</Text>
 <Object index="0"/>
 <Role index="child"/>
 <Object index="child"/>
 <Text>then this</Text>
 <Object index="1"  />
 <Role index="parent"/>
 <Text>that</Text>
 <Object index="parent"/>
</Constraint>

<Constraint xsi:type="Equality">
 <Text> -[Equality]  </Text>
 <Object index="0"/>
 <Role index="first"/>
 <Object index="first"/>
 <Text>if and only if</Text>
 <Text>this </Text>
 <Object index="0"/>
 <Role index="second"/>
 <Object index="second"/>
</Constraint>

<Constraint xsi:type="Equality FactType">
```

```
 <Text> -[Equality]  </Text>
 <Object index="0"/>
 <Role index="First"/>
 <Object index="First"/>
 <Text>if and only if</Text>
 <Text>this</Text>
 <Object index="1"/>
 <Role index="Second"/>
 <Text>that</Text>
 <Object index="Second"/>
</Constraint>

<Constraint xsi:type="Exclusion">
 <Text> -[Exclusion]  No</Text>
 <Object index="0"/>
 <Role index="first"/>
 <Object index="first"/>
 <Text>and also</Text>
 <Role index="second"/>
 <Object index="second"/>
</Constraint>

<Constraint xsi:type="Exclusion FactType">
 <Text> -[Exclusion]  No</Text>
 <Object index="0"/>
 <Role index="first"/>
 <Object index="first"/>
 <Text>and also</Text>
 <Role index="second"/>
 <Text>that</Text>
 <Object index="second"/>
</Constraint>

<Constraint xsi:type="Frequency">
 <Text> -[Frequency] If </Text>
 <Object index="0"/>
 <Role index="0"/>
 <Object index="1"/>
 <Role index="0"/>
 <Text>, then this </Text>
 <Object index="0"/>
 <Role index="0"/>
 <Text>at least </Text>
 <Minimum/>
 <Text> and most most </Text>
 <Maximum/>
 <Role index="0"/>
 <Text>(s)</Text>
</Constraint>

<Constraint xsi:type="Irreflexive">
```

```
 <Text> -[Irreflexive]  No</Text>
 <Object index="0"/>
 <Role index="0"/>
 <Text> it/him self</Text>
</Constraint>

<Constraint xsi:type="Symmetric">
 <Text> -[Symmetric]  If</Text>
 <Object index="0"/>
 <Text>X</Text>
 <Role index="0"/>
 <Object index="0"/>
 <Text>Y</Text>
 <Text>, it must be vice versa</Text>
</Constraint>

<Constraint xsi:type="Asymmetric">
 <Text> -[Symmetric]  If</Text>
 <Object index="0"/>
 <Text>X</Text>
 <Role index="0"/>
 <Object index="0"/>
<Text> Y, it cannot be be vice versa</Text>
</Constraint>

<Constraint xsi:type="Acyclic">
 <Text> -[Acyclic]  </Text>
 <Object index="0"/>
 <Text> cannot be directly (or indirectly through a chain)</Text>
 <Role index="0"/>
 <Text> it/him self</Text>
</Constraint>

<Constraint xsi:type="Transitve">
 <Text> -[Intransitve]  If</Text>
 <Object index="0"/>
 <Text>X</Text>
 <Role index="0"/>
 <Object index="0"/>
 <Text>Y, and Y</Text>
 <Role index="0"/>
 <Text> Z, then it cannot be that X</Text>
 <Role index="0"/>
 <Text>Z</Text>
</Constraint>

</ORMNLBody>
</ORMSchema>
```

-D

Example (Verbalizations in English)

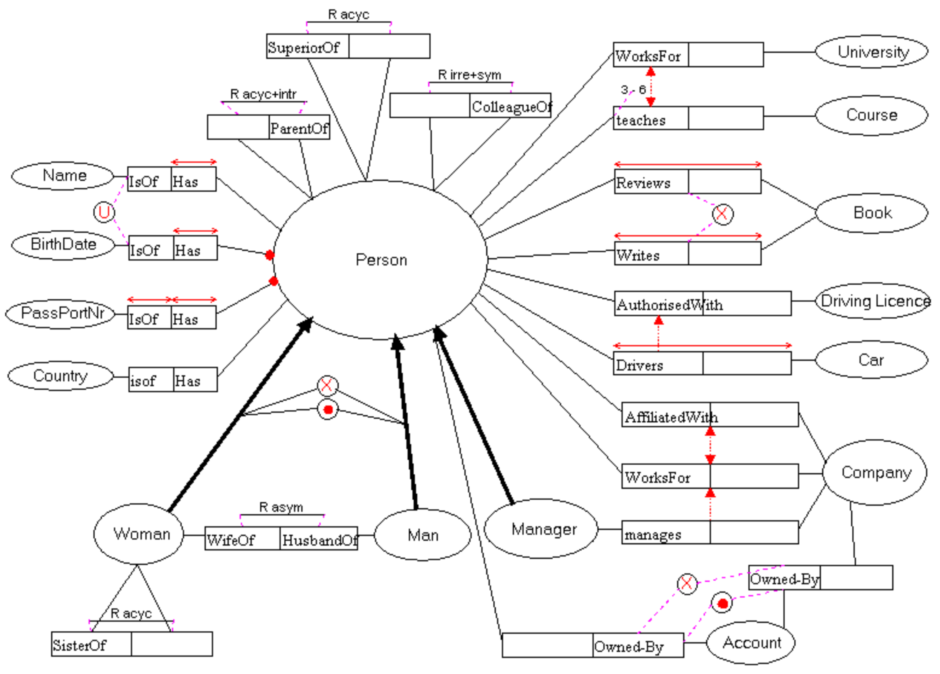


Fig. B.1. ORM-Diagram, English.

Verbalization

-[Mandatory] Each *Person* must Has at least one *PassPortNr*.
-[Mandatory] Each *Person* must Has at least one *BirthDate*.
-[Mandatory] Each *Account* should be *Owned-By Person* or *Owned-By Company*.
-[Uniqueness] Each *Person* must Has at most one *BirthDate*.
-[Uniqueness] Each *Person* must Has at most one *Name*.
-[Uniqueness] Each *Person* must Has at most one *PassPortNr*.
-[Uniqueness] Each *PassPortNr* must *IsOf* at most one *Person*.
-[Uniqueness] It is possible that *Person teaches* more than one *Course* , and vice versa.
-[Uniqueness] It is possible that *Person Reviews* more than one *Book* , and vice versa.
-[Uniqueness] It is possible that *Person Writes* more than one *Book* , and vice versa.
-[Uniqueness] It is possible that *Person Drivers* more than one *Car* , and vice versa.
-[Uniqueness] The combination of { *Name* and *BirthDate* } must refer to at most one *Person*.
-[Exlusive] Each *Person* should be either *Woman* or *Man*.
-[Totality] Each *Person* must be, at least, *Woman* or *Man*.
-[Subset] If *Person* *Drivers Car* then this *Person* *AuthorisedWith Driving Licence*.
-[Subset] If *Manager* manages *Company* then this *Person WorksFor* that *Company*.
-[Equality] *Person* WorksFor University if and only if this *Person teaches Course*.
-[Equality] *Person AffiliatedWith Company* if and only if this *Person WorksFor* that *Company*.
-[Exclusion] No *Account* *Owned-By Person* and also *Owned-By Company*.
-[Exclusion] No *Person Reviews Book* and also *Writes* that *Book*.

-D

-[Value] The possible instances of *Country* are :{ Belgium, France, Germany}
-[Irreflexive] No Person ColleagueOf it/him self.
-[Symmetric] If Person X ColleagueOf Person Y, it must be vice versa.
-[Acyclic] Person cannot be directly (or indirectly through a chain) ParentOf it/him self.
-[Acyclic] Person cannot be directly (or indirectly through a chain) SuperiorOf it/him self.
-[Asymmetric] If Person X WifeOf Person Y, it cannot be vice versa.
-[Intransitve] If Person X ParentOf Person Y, and Y ParentOf Z, then it cannot be that X ParentOf Z.
-[Frequency] If *Person teaches Course*, then this *Person teaches* at least 2 and most most 3 Course(s).

Dutch verbalization template

```
<?xml version='1.0' encoding='UTF-8'?>
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.jarrar.info'>

<ORMNLMeta>
 <Meta name="DC.Title" content="Dutch verbalization template (Ver0.3)"/>
 <Meta name="DC.Version" content="0.2"/>
 <Meta name="DC.Creator" content="Mustafa Jarrar"/>
   <Meta name="DC.Contributor" content="Pieter Verheyden"/>
 <Meta name="DC.Language" content="Dutch"/>
</ORMNLMeta>

<ORMNLBody>

<FactType xsi:type="FactType" >
<Text>Een</Text>
<Object index="0" />
<Role index="0"  />
<Text>/</Text>
<Role index="1"  />
<Text> een</Text>
<Object index="1" />
</FactType>

<Constraint xsi:type="Mandatory"  >
 <Text> -[Mandatory] E k(e)</Text>
 <Object index="0"  />
 <Role index="0"  />
 <Text> tenminste 1</Text>
 <Object index="1"  />
</Constraint>

<Constraint xsi:type="Backward Mandatory"  >
 <Text> -[Mandatory] Voor elk(e)</Text>
 <Object index="0"  />
<Text>is er tenminste 1</Text>
<Object index="1"  />
 <Text> dat</Text>
<Role index="1"  />
 <Text> dit/deze</Text>
 <Object index="0"  />
</Constraint>

<Constraint xsi:type="Disjunctive Mandatory" >
<Text> -[Mandatory] Elk(e)</Text>
```

-D

```
<Object index="0" />
<Text>ofwel</Text>
<Role index="0"   />
<Text>een</Text>
<Object index="1" />
<Loop index="1" >
  <Text>ofwel </Text>
  <Role index="n"   />
  <Text>een</Text>
  <Object index="1" />
</Loop>
</Constraint>

<Constraint xsi:type="Uniqueness" >
<Text> -[Uniqueness] Elk(e)</Text>
<Object index="0" />
<Role index="0"  />
<Text> ten hoogste 1 </Text>
<Object index="1" />
</Constraint>

<Constraint xsi:type="Backward Uniqueness"  >
 <Text> -[Uniqueness] Voor elke </Text>
<Object index="0" />
<Text>is er ten hoogste een </Text>
<Object index="1"  />
<Text> dat/die </Text>
<Role index="1"  />
<Text> dit/deze </Text>
<Object index="0"  />
</Constraint>

<Constraint xsi:type="External Uniqueness" >
<Text> -[Uniqueness] Elke combinatie van</Text>
<Object index="1" />
<Loop index="1">
  <Text>en</Text>
  <Object index="n" />
</Loop>
<Text>is gerelateerd met slechts 1</Text>
<Object index="0" />
</Constraint>

<Constraint xsi:type="Many Uniqueness" >
<Text> -[Uniqueness] Het is mogelijk dat een </Text>
<Object index="0" />
<Role index="0"></Role>
<Text>meer dan 1</Text>
<Object index="1" />
<Text>, en omgekeerd </Text>
</Constraint>

<Constraint xsi:type="Subtype" >
<Text> -[Subtype] Elk(e)</Text>
<Object index="child" />
<Text>is ook een</Text>
<Object index="parent"  />
</Constraint>

<Constraint xsi:type="Exclusive" >
<Text> -[Exclusive] E k(e)</Text>
```

```
<Object index="0"/>
<Text> kan ofwel een</Text>
<Object index="1"/>
<Loop index="1">
  <Text>ofwel een</Text>
  <Object index="n" />
</Loop>
<Text>zijn</Text>
</Constraint>

<Constraint xsi:type="Total" >
<Text> -[Total] Elk(e)</Text>
<Object index="0" />
<Text>is tenminste een</Text>
<Object index="1"  />
<Loop index="1" >
  <Text>of een</Text>
  <Object index="n" />
</Loop>
</Constraint>

<Constraint xsi:type="Subset" >
<Text> -[Subset] Als een</Text>
<Object  index="0" />
<Role index="child"  />
<Text>een</Text>
<Object index="child" />
<Text>dan moet ook  dit/deze</Text>
<Object index="1"  />
<Role index="parent"  />
<Text>een</Text>
<Object index="parent" />
</Constraint>

<Constraint xsi:type="Subset FactType" >
<Text> -[Subset] Als een  </Text>
<Object  index="0" />
<Role index="child"  />
<Text>een</Text>
<Object index="child" />
<Text>dan moet ook dit/deze </Text>
<Object index="1"  />
<Role index="parent"  />
<Text> dat </Text>
<Object index="parent" />
</Constraint>

<Constraint xsi:type="Equality" >
<Text> -[Equality] Een </Text>
<Object index="0"  />
<Role index="first"  />
<Text>een </Text>
<Object index="first"  />
<Text>dan en slechts dan als</Text>
<Text>dit/deze </Text>
<Object index="0" />
<Role index="second"  />
<Text>een </Text>
<Object index="second"  />
</Constraint>
```

```
<Constraint xsi:type="Equality FactType" >
<Text> -[Equality] Een</Text>
<Object index="0"  />
<Role index="0"  />
<Object index="1"  />
<Text>dan en slechts dan als</Text>
<Text>dit/deze </Text>
<Object index="0"  />
<Role index="1"  />
<Text>dat/die</Text>
<Object index="2"  />
</Constraint>

<Constraint xsi:type="Exclusion" >
<Text> -[Exclusion] Geen </Text>
<Object index="0"  />
<Role index="0"  />
<Text>een </Text>
<Object index="1"  />
<Text>en ook</Text>
<Role index="1"  />
<Text>Een </Text>
<Object index="2"  />
</Constraint>

<Constraint xsi:type="Exclusion FactType" >
<Text> -[Exclusion] Geen </Text>
<Object index="0"  />
<Role index="0"  />
<Text>een </Text>
<Object index="1"  />
<Text>en ook</Text>
<Role index="1"  />
<Text>datzelfde </Text>
<Object index="2"  />
</Constraint>

<Constraint xsi:type="Frequency">
 <Text> -[Frequency] If </Text>
 <Object index="0"/>
 <Role index="0"/>
 <Object index="1"/>
 <Role index="0"/>
 <Text>, then this </Text>
 <Object index="0"/>
 <Role index="0"/>
 <Text>at least </Text>
 <Minimum/>
 <Text> and most most </Text>
 <Maximum/>
 <Role index="0"/>
 <Text>(s)</Text>
</Constraint>

<Constraint xsi:type="Irreflexive">
 <Text> -[Irreflexive] Geen enkel(e)</Text>
 <Object index="0"/>
 <Role index="0"/>
 <Text> zichzelf/hemzelf</Text>
</Constraint>
```

-D

```
<Constraint xsi:type="Symmetric"  >
<Text>-[Symmetric] Indien</Text>
<Object index="0"/>
<Text> X</Text>
<Role index="0"/>
<Object index="0"/>
<Text> Y</Text>
<Text> , dan ook vice-versa</Text>
</Constraint>

<Constraint xsi:type="Asymmetric">
 <Text> -[Asymmetric] Indien</Text>
 <Object index="0"/>
 <Text> X</Text>
 <Role index="0"/>
 <Object index="0"/>
<Text> Y, dan kan het niet vice-versa</Text>
</Constraint>

<Constraint xsi:type="Acyclic">
 <Text> -[Acyclic]</Text>
 <Object index="0"/>
 <Text> kan niet rechtstreeks (of onrechtstreeks door een aaneenschakeling)</Text>
 <Role index="0"/>
 <Text> zichzelf/hemzelf</Text>
</Constraint>

<Constraint xsi:type="Transitve">
 <Text> -[Intransitve] Indien</Text>
 <Object index="0"/>
 <Text>X</Text>
 <Role index="0"/>
 <Object index="0"/>
 <Text>Y, en Y</Text>
 <Role index="0"/>
 <Text> Z, dan is het niet mogel jk dat X</Text>
 <Role index="0"/>
 <Text>Z</Text>
</Constraint>

</ORMNLBody>
</ORMSchema>
```

-D

Example (Verbalizations in Dutch)

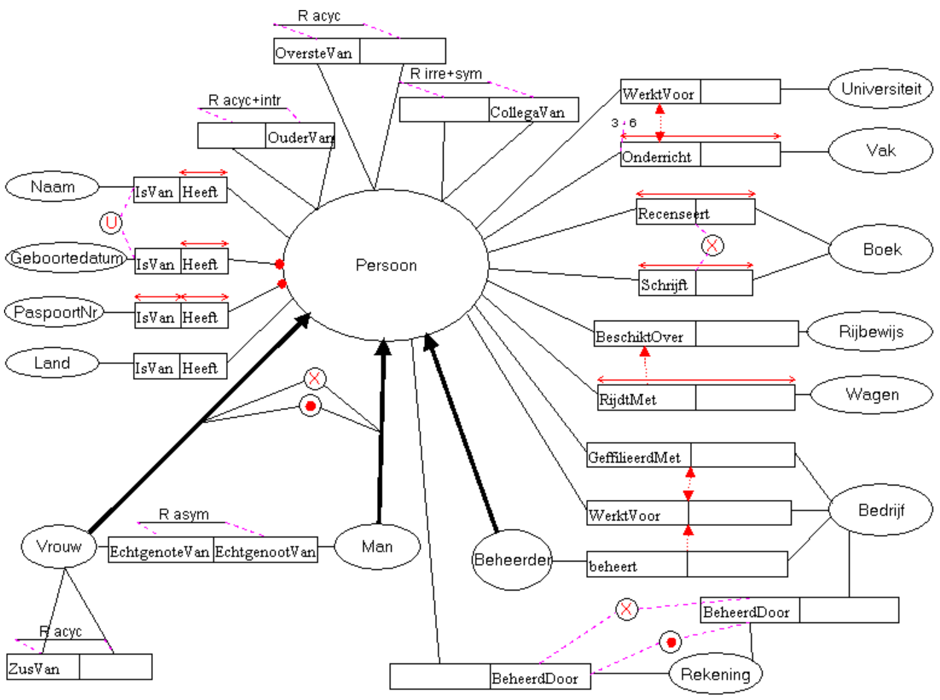


Fig. B.2. ORM-Diagram, Dutch.

Verbalization

-[Mandatory] Elk(e) Persoon Heeft tenminste 1 PaspoortNr.
-[Mandatory] Elk(e) Persoon Heeft tenminste 1 Geboortedatum.
-[Mandatory] Elk(e) Rekening ofwel BeheerdDoor een Persoon ofwel BeheerdDoor een Bedrijf.
-[Uniqueness] E k(e) Persoon Heeft ten hoogste 1 Geboortedatum.
-[Uniqueness] E k(e) Persoon Heeft ten hoogste 1 Naam.
-[Uniqueness] E k(e) Persoon Heeft ten hoogste 1 PaspoortNr.
-[Uniqueness] E k(e) PaspoortNr IsVan ten hoogste 1 Persoon.
-[Uniqueness] E ke combinatie van Naam en Geboortedatum is gerelateerd met slechts 1 Persoon.
-[Uniqueness] Het is mogel jk dat een Persoon Onderricht meer dan 1 Vak , en omgekeerd .
-[Uniqueness] Het is mogel jk dat een Persoon Recenseert meer dan 1 Boek , en omgekeerd .
-[Uniqueness] Het is mogel jk dat een Persoon Schr jft meer dan 1 Boek , en omgekeerd .
-[Uniqueness] Het is mogel jk dat een Persoon RijdtMet meer dan 1 Wagen , en omgekeerd .
-[Exclusive] Elk(e) Persoon kan ofwel een Man ofwel een Vrouw zijn.
-[Total] Elk(e) Persoon is tenminste een Vrouw of een Man.
-[Subset] Als een Persoon R jdtMet een Wagen dan moet ook dit/deze Persoon Besch ktOver een R jbewijs.
-[Subset] Als een Beheerder beheert een Bedr jf dan moet ook dit/deze Persoon WerktVoor dat Bedrijf.
-[Equality] Een Persoon WerktVoor een Universiteit dan en slechts dan als dit/deze Persoon

-D

Onderricht een Vak.
-[Equality] Een Persoon GeaffilieerdMet Bedr jf dan en slechts dan als dit/deze Persoon WerktVoor dat/die Bedrijf.
-[Exclusion] Geen enkel(e) Account Owned-By Person and also Owned-By Company.
-[Exclusion] No Person Reviews Book and also Writes that Book.
-[Value] De mogel jke instanties van Land zijn :{ Belgium, France, Germany}
-[Irreflexive] Geen enkel(e) Persoon CollegaVan zichzelf/hemzelf.
-[Symmetric] Indien Persoon X CollagaVan Persoon Y, dan ook vice-versa.
-[Acyclic] Persoon kan niet rechtstreeks (of onrechtstreeks door een aaneenschakeling) OversteVan zichzelf/hemzelf .
-[Acyclic] Vrouw kan niet rechtstreeks (of onrechtstreeks door een aaneenschakeling) ZusVan zichzelf/hemzelf .
-[Asymmetric] Indien Vrouw X EchtgenoteVanVrouw Y, dan kan het niet vice-versa .
-[Intransitve] Indien Persoon X OuderVan Persoon Y, en Y OuderVan Z, dan is het niet mogelijk dat X OuderVan Z.
-[Frequency] Indien Persoon Onderricht Vak, dan deze/dit Persoon Onderricht tenminste 2 en ten hoogste 3 Vak.

Acknowledgement: I am very grateful to Pieter Verheyden for his help in translating the Dutch verbalization template and the provided example.

-D

Arabic verbalization template

```xml
<?xml version='1.0' encoding='UTF-8'?>
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.jarrar.info/orm/verbalization/'>

<ORMNLMeta>
 <Meta name="DC.Title" content="Arabic verbalization template"/>
 <Meta name="DC.Version" content="0.2"/>
 <Meta name="DC.Creator" content="Mustafa Jarrar"/>
<Meta name="DC.Language" content="Arabic"/>
 </ORMNLMeta>
<ORMNLBody>

<FactType xsi:type="FactType" >
<Object index="0" />
<Role index="0"  />
<Text>/</Text>
<Role index="1"  />
<Object index="1" />
 </FactType>

<Constraint xsi:type="Mandatory"  >
 <Text> كل </Text>
 <Object index="0"  />
 <Role index="0"  />
 <Object index="1"  />
 <Text> واحد على الاقل</Text>
</Constraint>

<Constraint xsi:type="Backward Mandatory"  >
 <Text>لكل</Text>
 <Object index="0"  />
<Text>يوجد</Text>
<Object index="1"  />
<Text>واحد على الاقل</Text>
<Role index="1"  />
 <Text> هذا</Text>
 <Object index="0"  />
</Constraint>

<Constraint xsi:type="Disjunctive Mandatory">
 <Text> -[Mandatory]  كل </Text>
 <Object index="0"/>
 <Text>يجب ان يكون </Text>
 <Role index="0"/>
 <Object index="1"/>
 <Loop index="1" >
  <Text>او </Text>
  <Role index="n"/>
  <Object index="n"/>
 </Loop>
</Constraint>
```

```
<Constraint xsi:type="Uniqueness">
 <Text> -[Uniqueness]  لكل </Text>
 <Object index="0"/>
 <Role index="0"/>
 <Object index="1"/>
 <Text>واحد على الاكثر  </Text>
</Constraint>

<Constraint xsi:type="Backward Uniqueness"  >
<Text>لكل </Text>
<Object index="0" />
<Text> يوجد </Text>
<Object index="1"  />
<Text> واحد على الاكثر</Text>
<Role index="1"  />
 <Text> هذا</Text>
 <Object index="0"  />
</Constraint>

<Constraint xsi:type="Many Uniqueness" >
<Text>كل </Text>
<Object index="0" />
<Text>يمكن ان </Text>
<Role index="0"></Role>
<Text> اكثر من </Text>
<Object index="1"  />
<Text> والعكس صحيح </Text>
</Constraint>

<Constraint xsi:type="External Uniqueness" >
<Text>اتحاد كل من</Text>
<Object index="1"  />
<Loop index="1">
<Text>و</Text>
<Object index="n"  />
</Loop>
<Text>يشير الى</Text>
<Object index="0"   />
<Text> واحد على الاكثر</Text>
</Constraint>
<Constraint xsi:type="Subtype" >
<Text>كل</Text>
<Object index="child" />
<Text>هو</Text>
<Object index="parent"  />
 </Constraint>

<Constraint xsi:type="Value">
 <Text> -[Value]  اليقيم الممكنة ل </Text>
 <Object index="0"/>
 <Text>}:هي </Text>
 <Value index="0"/>
```

-D

```
 <Loop index="1">
  <Text>,</Text>
  <Value index="n"/>
 </Loop>
 <Text> {</Text>
 </Constraint>

<Constraint xsi:type="Subtype" >
  <Text>كل</Text>
  <Object index="child" />
  <Text>هو</Text>
  <Object index="parent"  />
</Constraint>

<Constraint xsi:type="Exclusive" >
<Text>كل</Text>
<Object index="0"/>
<Text> يمكن ان يكون اما</Text>
<Object index="1"/>
<Loop index="1">
  <Text>او</Text>
  <Object index="n"/>
</Loop>
 </Constraint>

<Constraint xsi:type="Total" >
 <Text>كل </Text>
<Object index="0" />
<Text>  يجب ان يكون </Text>
<Object index="1"  />
<Loop index="1" >
  <Text> او </Text>
  <Object index="n" />
</Loop>
</Constraint>

<Constraint xsi:type="Partition" >
<Text>كل </Text>
<Object index="0" />
<Text> يجب ان يكون اما </Text>
<Object index="1"  />
<Loop index="1" >
  <Text> او </Text>
  <Object index="n" />
</Loop>
</Constraint>

<Constraint xsi:type="Subset" >
<Text>اذا </Text>
<Object  index="0" />
<Role index="child"  />
<Object index="child" />
```

-D

```
<Text> فان هذا </Text>
<Object index="0"  />
<Role index="parent"  />
<Object index="parent" />
</Constraint>

<Constraint xsi:type="Subset FactType">
 <Text> -[Subset]  اذا </Text>
 <Object index="0"/>
 <Role index="child"/>
 <Object index="child"/>
 <Text> فان هذا  </Text>
 <Object index="1"  />
 <Role index="parent"/>
 <Text> هذة </Text>
 <Object index="parent"/>
</Constraint>

<Constraint xsi:type="Equality" >
<Text>كل </Text>
<Object index="0"  />
<Role index="first"  />
<Object index="first"  />
<Text>اذا و فقط اذاهذا ال </Text>
<Object index="0"  />
<Role index="second"  />
<Object index="second"  />
</Constraint>

<Constraint xsi:type="Equality FactType" >
<Text>كل </Text>
<Object index="0"  />
<Role index="0"  />
<Object index="1"  />
<Text>اذا و فقط اذاهذا ال</Text>
<Object index="1"  />
<Role index="second"  />
<Text>هذة ال</Text>
<Object index="second"  />
</Constraint>

<Constraint xsi:type="Exclusion" >
<Text>لا يمكن ان يكون  </Text>
<Object index="0"  />
<Role index="first"  />
<Object index="first"  />
<Text> و في نفس الوقت </Text>
<Role index="second"  />
<Object index="second"  />
</Constraint>

<Constraint xsi:type="Exclusion FactType" >
```

```
<Text>لا يمكن ان يكون </Text>
<Object index="0"  />
<Role index="first"  />
<Object index="first"  />
<Text>و في نفس الوقت </Text>
<Role index="second"  />
<Text>ذلك </Text>
<Object index="second"  />
 </Constraint>

<Constraint xsi:type="Frequency">
 <Text> -[Frequency]  اذا ال</Text>
 <Object index="0"/>
 <Role index="0"/>
 <Object index="1"/>
 <Role index="0"/>
 <Text>فان هذا ال </Text>
 <Object index="0"/>
 <Text>يجب ان </Text>
 <Role index="0"/>
 <Text>بين </Text>
 <Minimum/>
 <Text> الى </Text>
 <Maximum/>
 <Role index="0"/>
</Constraint>

<Constraint xsi:type="Irreflexive" >
<Text>لا يجوز ل </Text>
<Object index="0"/>
<Text> ان يكون </Text>
<Role index="0"/>
<Text> لنفسه </Text>
</Constraint>

<Constraint xsi:type="Symmetric"  >
<Text>اذا</Text>
<Object index="0"/>
<Text> س</Text>
<Role index="0"/>
<Object index="0"/>
<Text> ص</Text>
<Text> فانه العكس بالعكس</Text>
</Constraint>

<Constraint xsi:type="Asymmetric">
 <Text> -[Symmetric]  اذا</Text>
 <Object index="0"/>
 <Text>س </Text>
 <Role index="0"/>
 <Object index="0"/>
 <Text> </Text>
 <Text> ص, فان العكس غير صحيح </Text>
```

```
</Constraint>

<Constraint xsi:type="Acyclic">
 <Text> -[Acyclic]  لايمكن ل</Text>
 <Object index="0"/>
 <Text>ان يكون (بطريقة مباشرة او غير مباشرة) </Text>
 <Role index="0"/>
 <Text> نفسه</Text>
</Constraint>

<Constraint xsi:type="Transitve">
 <Text> -[Intransitve]  اذا </Text>
 <Object index="0"/>
 <Text>س  </Text>
 <Role index="0"/>
 <Object index="0"/>
 <Text> ص, و ص </Text>
 <Role index="0"/>
 <Text>ج, فانه لايمكن ان يكون س  </Text>
 <Role index="0"/>
 <Text>ج  </Text>
</Constraint>

</ORMNLBody>
</ORMSchema>
```

-D

Example (Verbalizations in Arabic)

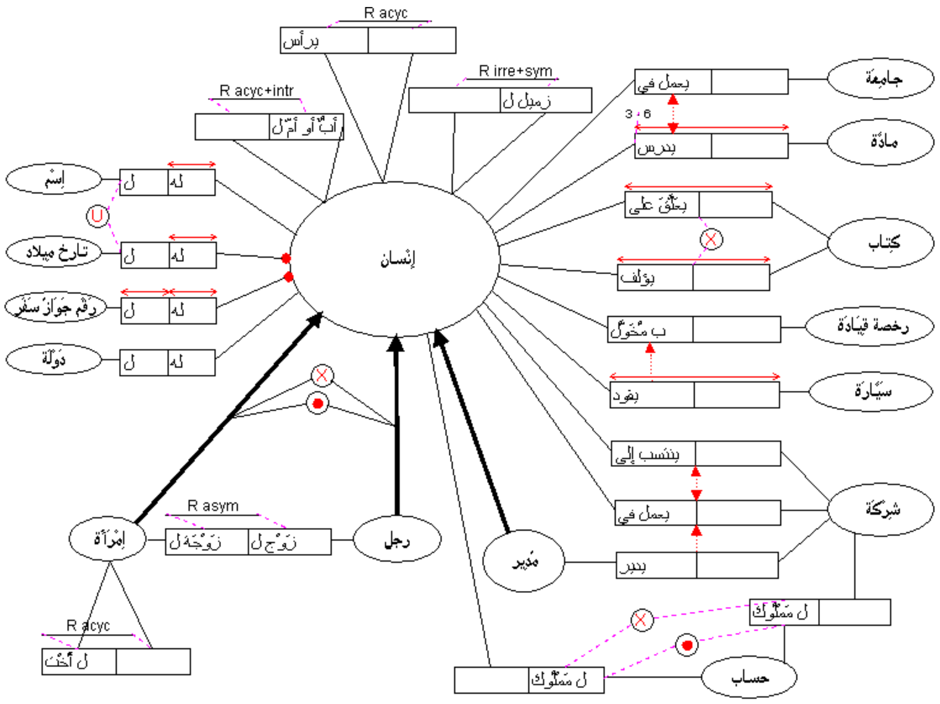


Fig. B.3. ORM-Diagram, Arabic.

Verbalization

-[Mandatory] كل إنْسان له رَقْم جَوَازُ سَفَر واحد على الاقل
-[Mandatory] كل إنْسان له تارخ مِيلاد واحد على الاقل
-[Mandatory] كل حساب يجب ان يكون مملوك ل انسان او مملوك ل شركة
-[Uniqueness] كل انسان له تاريخ ميلاد واحد على الاكثر
-[Uniqueness] كل انسان له اسم واحد على الاكثر
-[Uniqueness] كل انسان له رقم جواز سفر واحد على الاكثر
-[Uniqueness] كل رقم جواز سفر ل انسان واحد على الاكثر
-[Uniqueness] كل انسان يمكن ان يدرس اكثر من مادة والعكس صحيح
-[Uniqueness] كل انسان يمكن ان يؤلف اكثر من كتاب والعكس صحيح
-[Uniqueness] كل انسان يمكن ان يعلق على اكثر من كتاب والعكس صحيح
-[Uniqueness] كل انسان يمكن ان يقود اكثر من سيارة والعكس صحيح
-[Uniqueness] اتحاد كل من تاريخ ميلاد واسم يشير الى انسان واحد على الاكثر
-[Exclusive] كل انسان يمكن ان يكون اما رجل او اِمْرَأَة
-[Totality] كل انسان يجب ان يكون رجل او اِمْرَأَة
-[Subset] اذا انسان يقود سيارة فان هذا الانسان مخول ب رخصة سياقة
-[Subset] اذا مديريدير شركة فان هذا المديريعمل في هذة الشركة
-[Equality] كل انسان يعمل في جامعة اذا و فقط اذا هذا الانسان يدرس مادة

-D

-[Equality] كل انسان منسوب لشركة اذا و فقط اذا هذا الانسان يعمل في هذة الشركة
-[Exclusion] لا يمكن ان يكون حساب مملوك لا نسان و في نفس الوقت مملوك لشركة
-[Exclusion] لا يمكن ان يكون انسان يعلق على كتاب و في نفس الوقت يؤلف ذلك كتاب
-[Value] القيم الممكنة ل دولة هي:{ بلجيكا, فرنسا, المانيا}
-[Irreflexive] لا يجوز ل انسان ان يكون زميل لنفسه
-[Symmetric] اذا انسان س زميل ل ص,فانه العكس بالعكس
-[Acyclic] لايمكن لانسان ان يكون (بطريقة مباشرة او غير مباشرة) اب او ام لنفسه
-[Acyclic] لايمكن لانسان ان يكون (بطريقة مباشرة او غير مباشرة) مشرف على نفسه
-[Asymmetric] اذا انسان س زوجة لانسان ص, فان العكس غير صحيح
-[Intransitve] اذا انسان س اب او ام لانسان ص, و ص اب او ام لانسان ج, فانه لايمكن ان يكون س اب او ام ل ج
-[Frequency] اذا الانسان يدرس مادة, فان هذا الانسان يجب ان يدرس بين 2 الى 3 مادة

Russian verbalization template

```
<?xml version='1.0' encoding='UTF-8'?>
<ORMSchema xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://www.jarrar.info>

<ORMNLMeta>
 <Meta name="DC.Title" content="Russian verbalization template"/>
 <Meta name="DC.Version" content="0.1"/>
 <Meta name="DC.Creator" content="Mustafa Jarrar"/>
   <Meta name="DC.Contributor" content="Andriy Lisovoy"/>
 <Meta name="DC.Language" content="Russian"/>
</ORMNLMeta>
<ORMNLBody>

<Constraint xsi:type="Lexical"  >
<Text>Лексическими концепциями являются :{</Text>
 <Object index="0"  />
<Loop index="1">
<Text>,</Text>
 <Object index="n"  />
</Loop>
 <Text> }</Text>
 </Constraint>

<Constraint xsi:type="Value"  >
<Object index="0"  />
<Text> может быть представлен как :{</Text>
<Value index="0"  />
<Loop index="1">
<Text>,</Text>
 <Value index="n"  />
</Loop>
 <Text> }</Text>
 </Constraint>
```

-D

```
<Constraint xsi:type="Mandatory"  >
 <Text>Каждый</Text>
 <Object index="0"  />
 <Role index="0"  />
 <Text> по краней мере один</Text>
 <Object index="1"  />
</Constraint>

<Constraint xsi:type="Backward Mandatory"  >
 <Text>Для каждого</Text>
 <Object index="0"  />
<Text> существует по крайней мере один </Text>
<Object index="1"  />
 <Text> который </Text>
<Role index="1"  />
 <Text> этот</Text>
 <Object index="0"  />
</Constraint>

<Constraint xsi:type="Disjunctive Mandatory" >
<Object index="0" />
<Text>either</Text>
<Role index="0"   />
<Text>или</Text>
<Object index="1" />
<Loop index="1" >
 <Text>или </Text>
 <Role index="n"   />
<Object index="n" />
</Loop>
</Constraint>

<Constraint xsi:type="Uniqueness" >
<Text>Каждый</Text>
<Object index="0" />
<Role index="0"  />
<Text> максимум один </Text>
<Object index="1" />
 </Constraint>

<Constraint xsi:type="Backward Uniqueness"  >
<Text>Для каждого </Text>
<Object index="0" />
<Text> существует по максимум один </Text>
<Object index="1"  />
<Text> который </Text>
<Role index="1"  />
<Text> этот</Text>
<Object index="0"  />
</Constraint>
```

-D

```
<Constraint xsi:type="External Uniqueness" >
<Text>Каждая комбинация </Text>
<Object index="1"  />
<Loop index="1">
<Text>и</Text>
<Object index="n"  />
</Loop>
<Text> относится только к одному </Text>
<Object index="0"   />
</Constraint>

<Constraint xsi:type="Many Uniqueness" >
<Text>возможно, что</Text>
<Object index="0" />
<Role index="0"></Role>
<Text> больше, чем один </Text>
<Object index="1"  />
<Text> и, что</Text>
<Object index="1" />
<Role index="1"></Role>
<Text> больше, чем один </Text>
<Object index="0"  />
</Constraint>

<Constraint xsi:type="Subtype" >
<Text>Каждый</Text>
<Object index="child" />
<Text> также является </Text>
<Object index="parent"  />
</Constraint>

<Constraint xsi:type="Exclusive" >
<Text>Каждый</Text>
<Object index="0"/>
<Text> может быть  </Text>
<Object index="1"/>
<Loop index="1">
  <Text>или</Text>
  <Object index="n"/>
</Loop>
</Constraint>

<Constraint xsi:type="Total" >
 <Text>Каждый</Text>
<Object index="0" />
<Text> является либо </Text>
<Object index="1"  />
<Loop index="1" >
  <Text> или </Text>
  <Object index="n" />
```

```
</Loop>
</Constraint>

<Constraint xsi:type="Partition" >
<Text>Each </Text>
<Object index="0" />
<Text> по крайней мере является одним из </Text>
<Object index="1"  />
<Loop index="1" >
  <Text> или </Text>
  <Object index="n" />
</Loop>
<Text>но не всеми сразу</Text>
</Constraint>

<Constraint xsi:type="Subset" >
<Text>Если </Text>
<Object  index="0" />
<Role index="child"  />
<Object index="child" />
<Text>, то </Text>
<Object index="0"  />
<Role index="parent"  />
<Object index="parent" />
</Constraint>

<Constraint xsi:type="Equality" >
<Object index="0"  />
<Role index="first"  />
<Object index="first"  />
<Text>если и только если</Text>
<Text>этот </Text>
<Object index="0" />
<Role index="second"  />
<Object index="second"  />
<Text>, и наоборот</Text>
</Constraint>

<Constraint xsi:type="Equality FactType" >
<Object index="0"  />
<Role index="First"  />
<Object index="First"  />
<Text>если и только если</Text>
<Text>этот </Text>
<Object index="1"  />
<Role index="Second"  />
<Object index="Second"  />
</Constraint>

<Constraint xsi:type="Subset FactType" >
<Text>Если  </Text>
```

-D

```
<Object  index="0" />
<Role index="child"  />
<Object index="child" />
<Text>, то этот </Text>
<Object index="1"  />
<Role index="parent"  />
<Text> тот </Text>
<Object index="parent" />

</Constraint>

<Constraint xsi:type="Exclusion" >
<Text>Не существует </Text>
<Object index="0"  />
<Text>, который </Text>
<Role index="first"  />
<Object index="first"  />
<Text> и </Text>
<Role index="second"  />
<Object index="second"  />
</Constraint>

<Constraint xsi:type="Exclusion FactType" >
<Text>Не существует </Text>
<Object index="0"  />
<Text>, который </Text>
<Role index="first"  />
<Object index="first"  />
<Text>и</Text>
<Role index="second"  />
<Text>тот </Text>
<Object index="second"  />
 </Constraint>

<Constraint xsi:type="Reflexive" >
<Text>Каждый</Text>
<Object index="0"/>
<Role index="0"/>
<Text> </Text>
</Constraint>

<Constraint xsi:type="Irreflexive" >
<Text>No</Text>
<Object index="0"/>
<Role index="0"/>
<Text> самого себя</Text>
</Constraint>

<Constraint xsi:type="Symmetric"  >
<Text>Если</Text>
<Object index="0"/>
```

-D

```
<Text> x</Text>
<Role index="0"/>
<Object index="0"/>
<Text> y</Text>
<Text> то наоборот</Text>
</Constraint>

<Constraint xsi:type="Transitve"  >
<Text>Если</Text>
<Object index="0"/>
<Text>x</Text>
<Role index="0"/>
<Object index="0"/>
<Text>y и y</Text>
<Role index="0"/>
<Text> x то x</Text>
<Role index="0"/>
<Text>y</Text>
</Constraint>

</ORMNLBody>
</ORMSchema>
```

Example (Verbalizations in Russian)

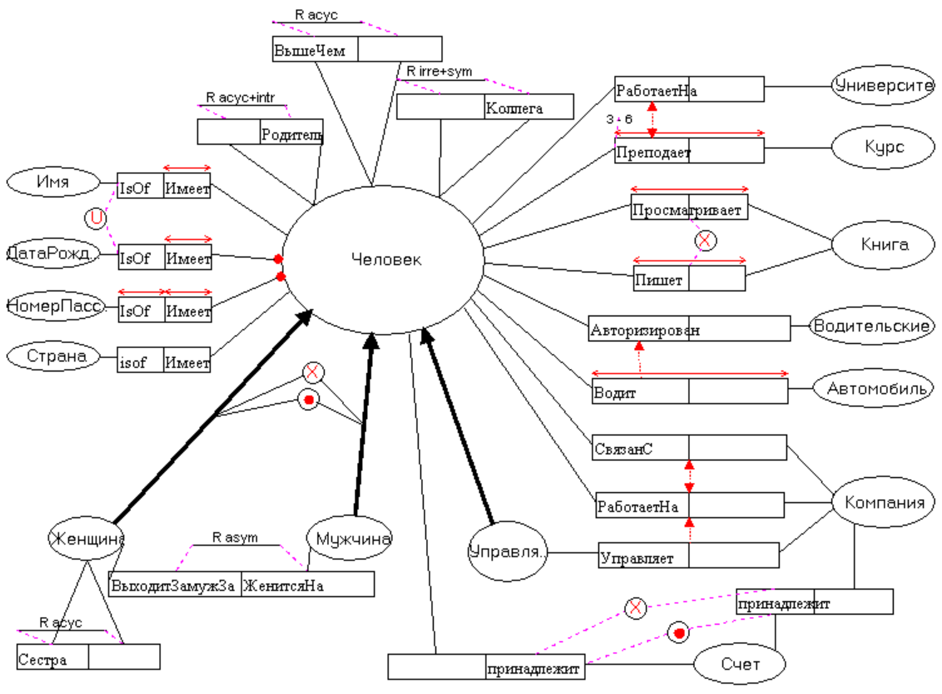


Fig. B.4. ORM-Diagram, Russian.

Verbalization

Каждый Человек Имеет по краней мере один НомерПасспорта.
Каждый Человек Имеет по краней мере один ДатаРождения.
Каждая комбинация ДатаРождения и Имя относится только к одному Человек.
Каждый Человек может быть Женщина или Мужчина.
Каждый Человек является либо Женщина или Мужчина.
Если Человек Водит Автомобиль , то Человек Авторизирован ВодительскиеПрава.
Человек РаботаетНаУниверситет если и только если этот Человек ПреподаетКурс , и наоборот.
Человек СвязанС Компания если и только если этот Человек РаботаетНа Компания.
Если Управляющий Управляет Компания , то этот Человек РаботаетНа тот Компания.
Не существует Счет , который принадлежитЧеловек и принадлежитКомпания.
Не существует Человек , который Пишет Книга и Просматривает тот Книга..
Счет either принадлежит или Человек или принадлежит Компания.
Каждый Человек Имеет максимум один ДатаРождения.
Каждый Человек Имеет максимум один Имя.
Каждый Человек Имеет максимум один НомерПасспорта.
Каждый НомерПасспорта IsOf максимум один Человек.
возможно, что Человек Преподает больше, чем один Курс и, что Курс Преподает больше,

чем один Человек.
возможно, что Человек Просматривает больше, чем один Книга и, что Книга Просматривает больше, чем один Человек.
возможно, что Человек Пишет больше, чем один Книга и, что Книга Пишет больше, чем один Человек.
возможно, что Человек Водит больше, чем один Автомобиль и, что Автомобиль Водит больше, чем один Человек.

Acknowledgement: I am very grateful to Andriy Lisovoy for his help in translating the Russian verbalization template and the provided example.

Appendix C: Customer Complaint Ontology

In this appendix, we present the CContology. In appendix C1, we present all terms and their glosses (CCglossary). The set of lexons are presented in appendix C2.

Appendix C1: The CCglossary

In this appendix, we present the CCglossary, which includes all terms and their glosses that have been used in the CContology. This CCglossary will be shared and used by people who wish to translate or extend the CContology.

Terms are listed in the alphabetical order.

Context	Term	Gloss
Customer Complaint	Access cost unreasonable	A private data access problem related to unreasonable access cost.
Customer Complaint	Access provision denied	A private data access problem related to denied access provision.
Customer Complaint	Access timeliness delayed	A private data access problem related to delayed access timeliness.
Customer Complaint	Action Request	An economic complaint resolution not related to financial issues, such as delivery, repair, etc.
Customer Complaint	Address	A construct describing the means by which contact may be taken with, or messages or physical objects may be delivered to; an address may contain indicators for a physical or virtual (i.e. accessed electronically) location or both.
Customer Complaint	Advance withheld	A contract termination problem related to advance payment was withheld unjustifiably at the termination of the contract, or not accounted properly against the payments during the contract.
Customer	Advertiser not	A advertising problem related to advertisements

-D

Complaint	identified	where the advertiser is not known or identified.
Customer Complaint	Advertising	Incorrect marketing practices problem related to advertisements of products or services.
Customer Complaint	After Sales Service Problem	A problem related to after sale service not actioned or not properly actioned.
Customer Complaint	Apartment Number	A number assigned to an apartment (flat/studio/office/room etc.) within a building.
Customer Complaint	Apologize	A symbolic resolution concerned with acknowledge faults, or shortcomings or failing.
Customer Complaint	Billing or Payment Problem	A purchase phase problem linked to billing or payment.
Customer Complaint	Billing Request	A financial request concerned with billing issues.
Customer Complaint	Breach of contract	A contract termination problem related to a breach of contract.
Customer Complaint	Building Name	A name assigned to a building or construction in or adjacent to which a delivery point is located.
Customer Complaint	Building Number	A number denoting a delivery point within a street; examples: house number, construction plot number.
Customer Complaint	Cancellation or withdrawal refused	A contract termination problem linked to a request of the consumer to withdraw from the contract is refused by the supplier.
Customer Complaint	Charge exceeds estimate	A repair problem related to charges exceeds the estimate.
Customer Complaint	City	(WordNet) An incorporated administrative district established by state charter.
Customer Complaint	Compensation inadequate	A guarantee problem related to inadequate compensation.
Customer Complaint	Compensation refused	A guarantee problem related to refusal of compensation.
Customer Complaint	Competitor cheaper	A competitor offers the same product or service at a lower price.
Customer Complaint	Complainant	The legal person who issues a complaint.
Customer Complaint	Complaint	An expression of grievance or resentment issued by a complainant against a compliant-recipient,

-D

		describing a problem(s) that needs to be resolved.
Customer Complaint	Complaint Date	The issue date of a complaint.
Customer Complaint	Complaint Number	A code used to uniquely refer to a complaint in a court or a complaint system.
Customer Complaint	Complaint Recipient	A legal person to whom a complaint is addressed.
Customer Complaint	Complaint Resolution	A determination for settling or solving a problem in a consumer-provider relationship.
Customer Complaint	Conduct	A non-problem problem concerned with the conduct of the recipient's staff, agents or sub-contractors.
Customer Complaint	Contact Details	A channel of communication
Customer Complaint	Content	A non-problem problem concerned with harmful or illegal content.
Customer Complaint	Contract	A binding agreement between two or more legal persons that is enforceable by law; an invoice can be a contract.
Customer Complaint	Contract Effective Date	The date on which the contract comes into effect, e.g. the date for the start of service.
Customer Complaint	Contract Order Date	The date on which the order was placed or the contract was signed.
Customer Complaint	Contract Problem	Problem linked to a contract in a customer-provider relationship, it may occar before or after the contract effective date.
Customer Complaint	Contract Reference	Reference to Contract, indicator to a certain contract
Customer Complaint	Contract rescinded	The recipient has rescinded the contract.
Customer Complaint	Contract Termination Problem	A problem concerned with the proper termination or completion of the contract.
Customer Complaint	Contract Terms Problem	A purchase phase problem linked to contracts terms and conditions.
Customer Complaint	Copyright	A non-contract problem concerned with exclusive and registered rights.
Customer Complaint	Country	(WordNet)The territory occupied by a nation.

Customer Complaint	County	(WordNet) A region created by territorial division for the purpose of local government
Customer Complaint	Damage	A non-contract problem concerned with damage suffered.
Customer Complaint	Damage Assessment	An action request concerned with judging or estimating a damage.
Customer Complaint	Data Collection	A privacy problem regarding all activities and purposes of private data collection
Customer Complaint	Data correction denied	Data correction was denied or executed incorrectly or delayed.
Customer Complaint	Data unrelated to purpose	A data collection problem concerned with Data unrelated to purpose in a customer-provider relationship.
Customer Complaint	Defective item not accepted for repair	A repair problem related to defective item not accepted for repair.
Customer Complaint	Delete the unnecessary data	A privacy request for delete private information specially that is unnecessary for the agreed purpose.
Customer Complaint	Delivery	The act of delivering or distributing goods or services.
Customer Complaint	Delivery and Installation Problem	A purchase phase problem related to dissatisfaction regarding delivery or Installation of goods or services.
Customer Complaint	Delivery charge problem	An unexpected delivery charge problem.
Customer Complaint	Delivery Consideration	Information denoted in a contract about a delivery agreements and circumstances, such as delivery address, date, loss or responsibility given, suffered or undertaken by the other.
Customer Complaint	Delivery problem	A purchase phase problem related to dissatisfaction regarding the delivery of goods or services.
Customer Complaint	Delivery Request	An action request concerned with delivery and distribution issues.
Customer Complaint	Deposit withheld	A contract termination problem linked with a deposit was withheld and not refunded.
Customer Complaint	Documentation in wrong language	The documentation or instructions were provided but are in the wrong language

-D

Customer Complaint	Documentation Problem	A product problem concerned with the product or service documentation or instructions.
Customer Complaint	Economic Resolution	A complaint resolution concerned with goods and services, such as payment, delivery, damage repair, etc.
Customer Complaint	Electronic Address	The address that can be accessed electronically (i.e. virtually), such as email, fax, pager, telephone, website, etc.
Customer Complaint	eMail	An electronic Address for transmission of letters and other documents from one computer to another through a telecommunications or wireless network.
Customer Complaint	Environmental damage	A damage problem related to environmental issues.
Customer Complaint	Evidence	(WordNet) all the means by which any alleged matter of fact whose truth is investigated at judicial trial is established or disproved
Customer Complaint	Excessive data requested	A data collection problem related to excessive data requested.
Customer Complaint	False statement	An advertising problem regarding a false (or not in accordance with the fact or reality or actuality) statement.
Customer Complaint	Fax	An electronic address used to transfer copies of documents, over a phone line.
Customer Complaint	Financial Reqeust	An economic complaint resolution concerned with financial issues, such as payments, billing, etc.
Customer Complaint	Function	The actions and activities assigned to or required or expected of one to play, such as sales agent, delivery driver, etc.
Customer Complaint	General Terms Problem	A contract terms problem with the general terms and conditions.
Customer Complaint	Gift defective or not received with product	A delivery problem regarding a gift defective or not received with product.
Customer Complaint	Goods	Durable or consumable articles of commerce including equipment, food, furniture, etc.
Customer Complaint	Guarantee Problem	An after sales service problem related to a legal or contractual guarantee; particularly a problem related to a responsibility on the recipient

-D

		consequent to the guarantees directive.
Customer Complaint	Guarantee refused	Refusal to apply a legal or contractual guarantee.
Customer Complaint	Harmful Content	A content problem related to harmful issues.
Customer Complaint	Hidden charges	A sales promotion problem regarding hidden charges.
Customer Complaint	High pressure selling	A sales methods problem concerned with using high pressure selling style.
Customer Complaint	Home selling problem	A personal selling problem regarding home selling practices.
Customer Complaint	Illegal Content	A content problem related to illegal content issues.
Customer Complaint	Illegal lottery	A sales promotion problem regarding illegal lottery.
Customer Complaint	Inadequate charge details	Details provided for a monetary charge are inadequate to identify that the charge is due.
Customer Complaint	Inadequate contact details	Details describing the contact details are inadequate to meet the requirements of European law, for example those required by the e-commerce directive or the data protection directive
Customer Complaint	Inadequate privacy information	The privacy information provided is inadequate/not compliant with legal requirements.
Customer Complaint	Inadequate specification	Specification of the product or service are not adequate for the complainant to make an informed purchasing decision.
Customer Complaint	Incorrect privacy information	A privacy information problem regarding the incorrectness of the privacy information.
Customer Complaint	Incorrect amount	An unexpected charge problem regarding incorrect amounts.
Customer Complaint	Incorrect assessment of a damage	A damage problem related to incorrect or not-acceptable assessment of damage.
Customer Complaint	Incorrect date	An unexpected charge problem regarding incorrect dates.
Customer Complaint	Incorrect interest charge	An unexpected charge problem regarding Incorrect interest charge.

Customer Complaint	Incorrect Marketing Practices	A pre-purchase problem related to marketing practices not in conformity with legal requirements.
Customer Complaint	Incorrect privacy information	A privacy information problem denoting incorrectness of information.
Customer Complaint	Incorrect quantity	A delivery problem of incorrect quantities.
Customer Complaint	Information Correction	A complaint resolution related to improvement to replace a mistake in the information collected in a consumer-provider relationship.
Customer Complaint	Information not comprehensible	An information problem linked to comprehensibility or understandability of Information.
Customer Complaint	Information not easily available	An information problem of not easily available.
Customer Complaint	Information Problem	A negotiation of terms problem related to information provided is incorrect, inadequate, or insufficient.
Customer Complaint	Installation delayed	An Installation problem related to delay in Installation.
Customer Complaint	Installation improper	An Installation problem denoting improper Installation.
Customer Complaint	Installation problem	A purchase phase problem related to dissatisfaction regarding the installation of goods or services.
Customer Complaint	Instructions inadequate	The instructions do not adequately indicate how some function works or some maintenance operation should be performed.
Customer Complaint	Instructions missing	Instructions for use or maintenance were not provided with the product.
Customer Complaint	Jurisdiction inappropriate	The jurisdiction specified is inappropriate because it is not aligned with the contract delivery or participants.
Customer Complaint	Legal information missing	An information problem denoting missing legal information.
Customer Complaint	Legal Person	#An entity with legal recognition in accordance with law, it has the legal capacity to represent its own interests in its own name, before a court of

-D

		law, to obtain rights or obligations for itself, to impose binding obligations, or to grant privileges
Customer Complaint	Lewd or Immoral conduct	A conduct problem related to Lewd and immoral issues.
Customer Complaint	Mailing Address	The address where a person or organization can be communicated with for providing physical objects. It is broadly equivalent to a postal address as described in standards CEN 14132 or UPU S42, but has different functional definition
Customer Complaint	Misleading advertising	An advertising problem regarding misleading advertisements.
Customer Complaint	Misrepresented needs	A repair problem related to misrepresented needs.
Customer Complaint	Money Request	A financial request concerned with money and currency issues, such as returning the money paid back, discount, etc.
Customer Complaint	Name	Name of a person (whether a natural or other legal person or a person without legal personality) to whom the contact details refer
Customer Complaint	Natural Person Complainant	A human being as distinguished from a person (as a corporation) created by operation of law, who issues a complaint.
Customer Complaint	Negotiation of Terms	A pre-purchase problem related to negotiation of the terms and conditions of a contract
Customer Complaint	No Discount	An unfair price problem related to not offering discounts.
Customer Complaint	No discount (usual one not offered)	An unfair price problem related to not offering discounts.
Customer Complaint	Non-Contract Problem	A Problem where there is no contract regarding a purchase in a customer-provider relationship.
Customer Complaint	Non-Natural Person Complainant	A legal person who is not a natural person (i.e. no a human being), and who issues a complaint. A non-natural person is also sometimes called "artificial person".
Customer Complaint	Not best offer	The contract is offered at a price that is not the best offer that the supplier is known to make in

		similar circumstances
Customer Complaint	Obtained data improperly	Some private data was obtained by improper/illegal means
Customer Complaint	Offensive	An advertising problem causing anger or annoyance because of violating or tending to violate or offend in advertisements.
Customer Complaint	Offer Problem	A negotiation of terms problem related to offer is not in compliance with legal requirements
Customer Complaint	Passed to an unauthorized country	A Purpose and permission privacy problem related to distributing private data to a country without authorization.
Customer Complaint	Passed to others without permission	A purpose and permission privacy problem related to distributing private data to others without permission or authority
Customer Complaint	Payment Consideration	Information denoted in a contract about a payment agreements and circumstances, such as, amounts, payment schedules, some right, interest, profit or benefit accruing to the one party suffered or undertaken by the other.
Customer Complaint	Payment details not provided	A payment problem related to no providing enough details about payment.
Customer Complaint	Payment Problem	A billing or payment problem related to dissatisfaction regarding payments.
Customer Complaint	Payment refused	A payment problem regarding to refusal of payment.
Customer Complaint	Personal selling	Incorrect marketing practices problem related to personal selling of products or services.
Customer Complaint	PO Box	A mailing address attribute denoting a designated box number for a delivery point provided by a postal operator; it may be provided for collection from a point operated by the postal operator or to facilitate bulk delivery to an organization.
Customer Complaint	Poor Advice	A personal selling problem related to poor advice.
Customer Complaint	Postal Code	(WordNet) A code of letters and digits added to a postal address to aid in the sorting of mail
Customer Complaint	PostalCode	A mailing address attribute denoting a code of letters and digits added to a postal address to aid in the sorting of mail

Customer Complaint	Post-purchase Phase Problem	A problem arising after a purchase.
Customer Complaint	Pre-purchase Phase Problem	A problem during the pre-contractual phase.
Customer Complaint	Price increase	An unexpected charge problem related to price increase.
Customer Complaint	Price unacceptable	Price is too high
Customer Complaint	Price Unfair	A contract terms problem related to price offered is not in accordance with price offered to other actual or potential purchasers; for example price is not in accordance with an advertised price.
Customer Complaint	Privacy Information	A privacy problem related to provision of private data
Customer Complaint	Privacy Problem	A problem related to the collection, storage, handling, use or distribution of private data, violating the data protection directives.
Customer Complaint	Privacy Request	A symbolic resolution related to the collection, storage, handling, use, distribution, access to or correction of private data.
Customer Complaint	Private Data Access	A privacy problem related to access and correction of private data
Customer Complaint	Prize not received	A sales promotion problem related to a prize no received.
Customer Complaint	Problem	A source of difficulty or dissatisfaction in a consumer-provider relationship.
Customer Complaint	Product delivery delayed	A delivery problem related to delay in product delivery.
Customer Complaint	Product fails standards compliance	A product quality (or delivery delayed) problem related to product fails standards compliance.
Customer Complaint	Product is defective	A product quality (or delivery delayed) problem related to Product defectiveness.
Customer Complaint	Product not delivered	A delivery problem regarding a product not delivered.
Customer Complaint	Product not in conformity to order	A delivery problem regarding a product not in conformity to order.
Customer	Product not	A delivery problem regarding a product not

Complaint	ordered	ordered.
Customer Complaint	Product performance below expectations	A product quality problem related to performance below expectations.
Customer Complaint	Product Problem	A problem linked a product provided by the provider.
Customer Complaint	Product Quality Problem	A product problem related to with the product quality.
Customer Complaint	Product unfit for purpose	A product delivery delayed problem related to unfit for purpose.
Customer Complaint	Product unsafe	A product quality problem related to product unsafe.
Customer Complaint	Property damage	A damage problem related to properties.
Customer Complaint	Provide access to the data	A privacy request of accessing the private data.
Customer Complaint	Provide the necessary privacy information	A privacy request of making the necessary privacy information and policies clearly visible.
Customer Complaint	Psychological damage	A damage problem related to psychological issues.
Customer Complaint	Purchase Phase Problem	A problem arising during the purchase phase.
Customer Complaint	Purpose and Permission	A privacy problem regarding access, collect, handle, distribute, etc. of private data without asking a permission or clarifying the purpose.
Customer Complaint	Receipt not confirmed	A payment problem regarding a receipt not confirmed.
Customer Complaint	Refund refused	A guarantee problem regarding a refused refund.
Customer Complaint	Refusal Problem	A negotiation of terms problem related to a provider refusing to take or cease some action which complainant could reasonably expect recipient to take.
Customer Complaint	Refusal to provide service	Recipient or another has refused to provide or continue to provide a services contracted directly or needed for another purchase, contract or guarantee to be effective..

Customer Complaint	Refusal to sell	Recipient or another has refused to sell goods or services to complainant or another
Customer Complaint	Registration	A certification, issued by an administrative authority or an accredited registration agency, declaring the official enrollment of an entity. Typically, it includes the official name, mailing address, registration number, VAT number, legal bases, etc.
Customer Complaint	Repair (ed item) not returned	A repair problem regarding a repair (ed item) not returned.
Customer Complaint	Repair delayed	The repair time, either delivered or proposed, is too long
Customer Complaint	Repair inadequate	The repair made was inadequate
Customer Complaint	Repair Problem	An after sales service problem related to a repair.
Customer Complaint	Repair refused	A repair under guarantee was refused
Customer Complaint	Replacement refused	A replacement under guarantee was refused
Customer Complaint	Reputation damage	A damage problem related to reputation, esteem, and honor of people and institutions.
Customer Complaint	Right to object denied	A private data access problem regarding a denied right to object.
Customer Complaint	Rights infringed	The legal or moral rights ofa party have been infringed
Customer Complaint	Rudeness	A conduct problem related to rudeness in a customer-provider relationship and communication.
Customer Complaint	Sales and contract Request	An action request concerned agreements and contract issues.
Customer Complaint	Sales Contact Method	A method by which one is contacted with respect to an actual or potential purchase or contract; examples: shop, direct mail, e-mail, web site, direct response advertisement, telephone, fax, door step, in the street.
Customer Complaint	Sales Methods	A non-contract problem concerned with sales methods

Customer Complaint	Sales Office	Location where the staff responsible for the sale or contract are normally working or to which they report; examples: shop, branch, field sales office , etc.
Customer Complaint	Sales promotion	Incorrect marketing practices problem related to promotions of products or services.
Customer Complaint	Schedule	(WordNet) An ordered list of times at which things are planned to occur.
Customer Complaint	Secondary purpose permission refusal denies primary service	A purpose an permission problem regarding secondary purpose permission refusal denies primary.
Customer Complaint	Service cancelled by provider	A service problem regarding a service cancelled by provider.
Customer Complaint	Service inadequately per-formed	A service problem regarding a service inadequately per-formed.
Customer Complaint	Service not ordered	A service problem regarding a service not ordered.
Customer Complaint	Service not provided	A service problem regarding a service not provided.
Customer Complaint	Service partially provided	A service problem regarding a service not partially provided.
Customer Complaint	Service Problem	An after sales service problem related to provision of a service.
Customer Complaint	Service provision delayed	A service problem regarding a delayed service provision.
Customer Complaint	Services	A commercial work done by one that benefits another.
Customer Complaint	Spare part not available	A repair problem related to a spare part not available.
Customer Complaint	Specification not adequate	Specification of the product or service are not adequate for the complainant to make an informed purchasing decision
Customer Complaint	State	(WordNet) The territory occupied by one of the constituent administrative districts of a nation
Customer	Stop	A privacy request to stop collecting, storing,

Complaint	processing and transmission of private data	handling, distributing, publishing, accessing, etc. of private data.
Customer Complaint	Street	(WordNet) A thoroughfare (usually including pavements) that is lined with buildings
Customer Complaint	Street selling problem	A personal problem regarding street selling.
Customer Complaint	Supplementary (charge problem)	An unexpected charge problem related to supplementary charges.
Customer Complaint	Switching or Churning	A contract termination problem
Customer Complaint	Symbolic Resolution	A complaint resolution concerned with emotional, moral, social, or privacy issues. Such as apology, provide access, stop processing, etc.
Customer Complaint	Telephone	An electronic address used for transmitting and receiving voice-frequency signals at a distance.
Customer Complaint	Terms and Conditions	The financial and management conditions under which venture capital limited partnerships are structured.
Customer Complaint	Terms modified	The terms and conditions have been modified without agreement
Customer Complaint	Third Party	(WordNet) Someone other than the principals who are involved in a transaction.
Customer Complaint	Third Party Name	The name of a third party.
Customer Complaint	Time Limit	An offer problem denoting too short time limits.
Customer Complaint	Time limit (too short)	Limit in time duration or date imposed by contract or mandated by law; for example the time limit available for repudiation of a contract made under conditions of the distance selling directive;
Customer Complaint	Total Amount Asked	The total of all amounts asked of the purchaser by the seller.
Customer Complaint	Total Amount Paid	The total of all amounts paid by the purchaser to the seller.
Customer Complaint	Trying to obtain data improperly	Some attempt was improperly made to acquire some personal data
Customer	Unacceptable	The contract terms offered are unacceptable

Complaint	terms	
Customer Complaint	unauthorized comparative advertising	An advertising problem related to unauthorized comparative advertising.
Customer Complaint	unauthorized repair	A repair problem related to unauthorized repair issues.
Customer Complaint	Unexpected charge	A billing or payment problem related to dissatisfaction regarding unexpected charge.
Customer Complaint	Unfair contest	A sales promotion problem related to unfair contests.
Customer Complaint	Unfair Contract Terms	A contractual term which has not been individually negotiated and causes a significant imbalance in the parties rights and obligations arising under the contract, to the detriment of the consumer..
Customer Complaint	Unfair packaging	A sales promotion problem related to unfair packaging.
Customer Complaint	Unjustified payment demand	An unexpected charge problem related to unjustified payment demand.
Customer Complaint	Unnecessary Purpose	A purpose and permission problem denoting unnecessary purpose.
Customer Complaint	Unproven health claim	An advertising problem related to unproven health claim.
Customer Complaint	Unsolicited commercial communication s	A sales methods problem concerned with unsolicited commercial communications
Customer Complaint	Unsolicited merchandise	A sales methods problem concerned with unsolicited merchandises.
Customer Complaint	Unsolicited service	A sales methods problem concerned with unsolicited services.
Customer Complaint	Untruthlness	A conduct problem to related to untruthlness in a customer-provider relationship and communication.
Customer Complaint	Used for purpose without permission	The personal data was used for some purpose for which permission was denied or withdrawn
Customer Complaint	Web Site	An electronic address on the World Wide Web network; normally formatted as a URL (universal

		resource locator) describing a virtual or physical web server, often a host name referenced within the domain name system (e.g. http://www.ccform.org)
Customer Complaint	Wrong Language	A documentation problem regarding the language of the attached documentations.

Appendix C2: Lexons in the CContology

In this appendix, we present the set of lexons in the CContology. Lexons are presented in the alphabetical ordered of Term$_1$.

Context	Term1	Role	InvRole	Term2
Customer Complaint	Action Request	Types	Subtypeof	Delivery Request
Customer Complaint	Action Request	Types	Subtypeof	Sales and contract Request
Customer Complaint	Action Request	Types	Subtype-Of	Damage Assessment
Customer Complaint	Address	Types	Subtype-Of	Electronic Address
Customer Complaint	Address	Types	Subtype-Of	Mailing Address
Customer Complaint	Advertising	Types	Subtype-Of	Advertiser not identified
Customer Complaint	Advertising	Types	Subtype-Of	False statement
Customer Complaint	Advertising	Types	Subtype-Of	Misleading advertising
Customer Complaint	Advertising	Types	Subtype-Of	Offensive
Customer Complaint	Advertising	Types	Subtype-Of	Unauthorized comparative advertising
Customer Complaint	Advertising	Types	Subtype-Of	Unproven health claim
Customer Complaint	After Sales Service Problem	Types	Subtype-Of	Guarantee Problem
Customer Complaint	After Sales Service Problem	Types	Subtype-Of	Repair Problem
Customer Complaint	After Sales Service	Types	Subtype-Of	Service Problem

-D

	Problem			
Customer Complaint	Billing or Payment Problem	Types	Subtype-Of	Payment Problem
Customer Complaint	Billing or Payment Problem	Types	Subtype-Of	Unexpected charge
Customer Complaint	Complainant	Types	Subtype-Of	Natural Person Complainant
Customer Complaint	Complainant	Types	Subtype-Of	Non-Natural Person Complainant
Customer Complaint	Complaint	against	receives	Complaint Recipient
Customer Complaint	Complaint	describes	described_by	Problem
Customer Complaint	Complaint	Has	is-of	Complaint Date
Customer Complaint	Complaint	Has	is-of	Complaint Number
Customer Complaint	Complaint	issued_by	issues	Complainant
Customer Complaint	Complaint	requests	requested_by	Complaint Resolution
Customer Complaint	Complaint Resolution	denoted_by	denotes	Contact Details
Customer Complaint	Complaint Resolution	denoted_by	denotes	Registration
Customer Complaint	Complaint Resolution	Types	Subtype-Of	Economic Resolution
Customer Complaint	Complaint Resolution	Types	Subtype-Of	Information Correction
Customer Complaint	Complaint Resolution	Types	Subtype-Of	Symbolic Resolution
Customer Complaint	Conduct	Types	Subtype-Of	Lewd or Immoral conduct
Customer Complaint	Conduct	Types	Subtype-Of	Rudeness
Customer	Conduct	Types	Subtype-Of	Untruthlness

-D

Complaint				
Customer Complaint	Contact Details	comprised_of	comprises	Address
Customer Complaint	Contact Details	Has	is-of	Name
Customer Complaint	Content	Types	Subtype-Of	Harmful Content
Customer Complaint	Content	Types	Subtype-Of	Illegal Content
Customer Complaint	Contract	Has	-	Contract Order Date
Customer Complaint	Contract	Has	-	Contract Effective Date
Customer Complaint	Contract	Has	is-of	Contract Reference
Customer Complaint	Contract	Has	is-of	Sales Contact Method
Customer Complaint	Contract	Has	is-of	Sales Office
Customer Complaint	Contract	Has	is-of	Terms and Conditions
Customer Complaint	Contract	involves	involved_in	Third Party
Customer Complaint	Contract	reports	-	Payment Consideration
Customer Complaint	Contract	reports	-	Delivery Consideration
Customer Complaint	Contract Problem	Types	Subtype-Of	Post-purchase Phase Problem
Customer Complaint	Contract Problem	Types	Subtype-Of	Pre-purchase Phase Problem
Customer Complaint	Contract Problem	Types	Subtype-Of	Purchase Phase Problem
Customer Complaint	Contract Termination Problem	Types	Subtype-Of	Advance withheld
Customer Complaint	Contract Termination Problem	Types	Subtype-Of	Breach of contract
Customer	Contract	Types	Subtype-Of	Cancellation or

Complaint	Termination Problem			withdrawal refused
Customer Complaint	Contract Termination Problem	Types	Subtype-Of	Deposit withheld
Customer Complaint	Contract Termination Problem	Types	Subtype-Of	Switching or Churning
Customer Complaint	Contract Terms Problem	Types	Subtype-Of	General Terms Problem
Customer Complaint	Contract Terms Problem	Types	Subtype-Of	Price Unfair
Customer Complaint	Damage	Types	Subtype-Of	Environmental damage
Customer Complaint	Damage	Types	Subtype-Of	Incorrect assessment of a damage
Customer Complaint	Damage	Types	Subtype-Of	Property damage
Customer Complaint	Damage	Types	Subtype-Of	Psychological damage
Customer Complaint	Damage	Types	Subtype-Of	Reputation damage
Customer Complaint	Data Collection	Types	Subtype-Of	Data unrelated to purpose
Customer Complaint	Data Collection	Types	Subtype-Of	Excessive data requested
Customer Complaint	Data Collection	Types	Subtype-Of	Obtained data improperly
Customer Complaint	Data Collection	Types	Subtype-Of	Trying to obtain data improperly
Customer Complaint	Delivery	Considered_b y	Considers	Delivery Consideration
Customer Complaint	Delivery	Has	is-of	Address
Customer Complaint	Delivery	Has	is-of	Goods
Customer	Delivery	Has	is-of	Schedule

-D

Complaint				
Customer Complaint	Delivery	Has	is-of	Services
Customer Complaint	Delivery and Installation Problem	Types	Subtype-Of	Delivery problem
Customer Complaint	Delivery and Installation Problem	Types	Subtype-Of	Installation problem
Customer Complaint	Delivery problem	Types	Subtype-Of	Gift defective or not received with product
Customer Complaint	Delivery problem	Types	Subtype-Of	Incorrect quantity
Customer Complaint	Delivery problem	Types	Subtype-Of	Product delivery delayed
Customer Complaint	Delivery problem	Types	Subtype-Of	Product not delivered
Customer Complaint	Delivery problem	Types	Subtype-Of	Product not in conformity to order
Customer Complaint	Delivery problem	Types	Subtype-Of	Product not ordered
Customer Complaint	Documentati on Problem	Types	Subtype-Of	Instructions inadequate
Customer Complaint	Documentati on Problem	Types	Subtype-Of	Instructions missing
Customer Complaint	Documentati on Problem	Types	Subtype-Of	Wrong Language
Customer Complaint	Economic Resolution	types	subtypeof	Financial Reqeust
Customer Complaint	Economic Resolution	Types	Subtype-Of	Action Request
Customer Complaint	Electronic Address	Types	Subtype-Of	eMail
Customer Complaint	Electronic Address	Types	Subtype-Of	Fax
Customer Complaint	Electronic Address	Types	Subtype-Of	Telephone

-D

Customer Complaint	Electronic Address	Types	Subtype-Of	Web Site
Customer Complaint	Financial Reqeust	types	subtypeof	Billing Request
Customer Complaint	Financial Reqeust	types	subtypeof	Money Request
Customer Complaint	General Terms Problem	Types	Subtype-Of	Contract Rescinded
Customer Complaint	General Terms Problem	Types	Subtype-Of	Jurisdiction inappropriate
Customer Complaint	General Terms Problem	Types	Subtype-Of	Rights Infringed
Customer Complaint	General Terms Problem	Types	Subtype-Of	Terms Modified
Customer Complaint	General Terms Problem	Types	Subtype-Of	Unfair Contract Terms
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Compensation inadequate
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Compensation Refused
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Guarantee Refused
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Refund Refused
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Repair Refused
Customer Complaint	Guarantee Problem	Types	Subtype-Of	Replacement Refused
Customer Complaint	Incorrect Marketing Practices	Types	Subtype-Of	Advertising
Customer Complaint	Incorrect Marketing Practices	Types	Subtype-Of	Personal selling
Customer	Incorrect	Types	Subtype-Of	Sales

-D

Complaint	Marketing Practices			promotion
Customer Complaint	Information Problem	Types	Subtype-Of	Inadequate Charge Details
Customer Complaint	Information Problem	Types	Subtype-Of	Inadequate Contact Details
Customer Complaint	Information Problem	Types	Subtype-Of	Inadequate Specification
Customer Complaint	Information Problem	Types	Subtype-Of	Information not comprehensibl e
Customer Complaint	Information Problem	Types	Subtype-Of	Information not easily available
Customer Complaint	Information Problem	Types	Subtype-Of	Legal information missing
Customer Complaint	Installation problem	Types	Subtype-Of	Installation delayed
Customer Complaint	Installation problem	Types	Subtype-Of	Installation improper
Customer Complaint	Legal Person	Types	Subtype-Of	Complainant
Customer Complaint	Legal Person	Types	Subtype-Of	Complaint Recipient
Customer Complaint	Mailing Address	Has	is-of	Apartment Number
Customer Complaint	Mailing Address	Has	is-of	Building Name
Customer Complaint	Mailing Address	Has	is-of	Building Number
Customer Complaint	Mailing Address	Has	is-of	City
Customer Complaint	Mailing Address	Has	is-of	Country
Customer Complaint	Mailing Address	Has	is-of	County
Customer Complaint	Mailing Address	Has	is-of	PO Box
Customer Complaint	Mailing Address	Has	is-of	PostalCode

Customer Complaint	Mailing Address	Has	is-of	State
Customer Complaint	Mailing Address	Has	is-of	Street
Customer Complaint	Natural Person Complainant	denoted_by	denotes	Registration
Customer Complaint	Negotiation of Terms	Types	Subtype-Of	Information Problem
Customer Complaint	Negotiation of Terms	Types	Subtype-Of	Offer Problem
Customer Complaint	Negotiation of Terms	Types	Subtype-Of	Refusal Problem
Customer Complaint	Non-Contract Problem	Types	Subtype-Of	Conduct
Customer Complaint	Non-Contract Problem	Types	Subtype-Of	Content
Customer Complaint	Non-Contract Problem	Types	Subtype-Of	Copyright
Customer Complaint	Non-Contract Problem	Types	Subtype-Of	Damage
Customer Complaint	Non-Contract Problem	Types	Subtype-Of	Sales Methods
Customer Complaint	Non-Natural Person Complainant	denoted_by	denotes	Contact Details
Customer Complaint	Offer Problem	Types	Subtype-Of	Price Unacceptable
Customer Complaint	Offer Problem	Types	Subtype-Of	Specification not Adequate
Customer Complaint	Offer Problem	Types	Subtype-Of	Time Limit
Customer Complaint	Offer Problem	Types	Subtype-Of	Unacceptable Terms
Customer Complaint	Offer Problem	Types	Subtype-Of	Unfair Contract Terms
Customer Complaint	Payment Consideratio n	Has	is-of	Total Amount Asked
Customer	Payment	Has	is-of	Total Amount

-D

Complaint	Consideratio n			Paid
Customer Complaint	Payment Problem	Types	Subtype-Of	Payment details not provided
Customer Complaint	Payment Problem	Types	Subtype-Of	Payment refused
Customer Complaint	Payment Problem	Types	Subtype-Of	Receipt not confirmed
Customer Complaint	Personal selling	Types	Subtype-Of	Home selling problem
Customer Complaint	Personal selling	Types	Subtype-Of	Poor Advice
Customer Complaint	Personal selling	Types	Subtype-Of	Street selling problem
Customer Complaint	Post-purchase Phase Problem	refers_to	Associated_wit h	Contract
Customer Complaint	Post-purchase Phase Problem	Types	Subtype-Of	After Sales Service Problem
Customer Complaint	Post-purchase Phase Problem	Types	Subtype-Of	Contract Termination Problem
Customer Complaint	Post-purchase Phase Problem	Types	Subtype-Of	Product Problem
Customer Complaint	Pre-purchase Phase Problem	Types	Subtype-Of	Incorrect Marketing Practices
Customer Complaint	Pre-purchase Phase Problem	Types	Subtype-Of	Negotiation of Terms
Customer Complaint	Price Unfair	Types	Subtype-Of	Competitor Cheaper
Customer	Price Unfair	Types	Subtype-Of	No Discount

Complaint				
Customer Complaint	Price Unfair	Types	Subtype-Of	Not Best Offer
Customer Complaint	Privacy Information	Types	Subtype-Of	Inadequate privacy information
Customer Complaint	Privacy Information	Types	Subtype-Of	Incorrect privacy information
Customer Complaint	Privacy Problem	Types	Subtype-Of	Data Collection
Customer Complaint	Privacy Problem	Types	Subtype-Of	Privacy Information
Customer Complaint	Privacy Problem	Types	Subtype-Of	Private Data Access
Customer Complaint	Privacy Problem	Types	Subtype-Of	Purpose and Permission
Customer Complaint	Privacy Request	Types	Subtype-Of	Delete the unnecessary data
Customer Complaint	Privacy Request	Types	Subtype-Of	Provide access to the data
Customer Complaint	Privacy Request	Types	Subtype-Of	Provide the necessary privacy information
Customer Complaint	Privacy Request	Types	Subtype-Of	Stop Processing and transmission of private data
Customer Complaint	Private Data Access	Types	Subtype-Of	Access cost unreasonable
Customer Complaint	Private Data Access	Types	Subtype-Of	Access provision denied
Customer Complaint	Private Data Access	Types	Subtype-Of	Access timeliness delayed
Customer Complaint	Private Data Access	Types	Subtype-Of	Data correction denied

-D

Customer Complaint	Private Data Access	Types	Subtype-Of	Right to object denied
Customer Complaint	Problem	testified_by	-	Evidence
Customer Complaint	Problem	Types	Subtype-Of	Contract Problem
Customer Complaint	Problem	Types	Subtype-Of	Non-Contract Problem
Customer Complaint	Problem	Types	Subtype-Of	Privacy Problem
Customer Complaint	Product delivery delayed	Types	Subtype-Of	Product fails standards compliance
Customer Complaint	Product delivery delayed	Types	Subtype-Of	Product is defective
Customer Complaint	Product delivery delayed	Types	Subtype-Of	Product performance below expectations
Customer Complaint	Product delivery delayed	Types	Subtype-Of	Product unfit for purpose
Customer Complaint	Product delivery delayed	Types	Subtype-Of	Product unsafe
Customer Complaint	Product Problem	Types	Subtype-Of	Documentation Problem
Customer Complaint	Product Problem	Types	Subtype-Of	Product Quality Problem
Customer Complaint	Product Quality Problem	Types	Subtype-Of	Product fails standards compliance
Customer Complaint	Product Quality Problem	Types	Subtype-Of	Product is defective
Customer Complaint	Product Quality Problem	Types	Subtype-Of	Product performance below expectations

Customer Complaint	Product Quality Problem	Types	Subtype-Of	Product unfit for purpose
Customer Complaint	Product Quality Problem	Types	Subtype-Of	Product unsafe
Customer Complaint	Purchase Phase Problem	refers_to	Associated_wit h	Contract
Customer Complaint	Purchase Phase Problem	Types	Subtype-Of	Billing or Payment Problem
Customer Complaint	Purchase Phase Problem	Types	Subtype-Of	Contract Terms Problem
Customer Complaint	Purchase Phase Problem	Types	Subtype-Of	Delivery and Installation Problem
Customer Complaint	Purpose and Permission	Types	Subtype-Of	Passed to an unauthorized country
Customer Complaint	Purpose and Permission	Types	Subtype-Of	Passed to others without permission
Customer Complaint	Purpose and Permission	Types	Subtype-Of	Secondary purpose permission refusal denies primary service
Customer Complaint	Purpose and Permission	Types	Subtype-Of	Unnecessary Purpose
Customer Complaint	Purpose and Permission	Types	Subtype-Of	Used for purpose without permission
Customer Complaint	Refusal Problem	Types	Subtype-Of	Refusal to Provide Service
Customer Complaint	Refusal Problem	Types	Subtype-Of	Refusal to Sell
Customer	Repair	Types	Subtype-Of	Charge

Complaint	Problem			exceeds estimate
Customer Complaint	Repair Problem	Types	Subtype-Of	Defective item not accepted for repair
Customer Complaint	Repair Problem	Types	Subtype-Of	Misrepresented needs
Customer Complaint	Repair Problem	Types	Subtype-Of	Repair (ed item) not returned
Customer Complaint	Repair Problem	Types	Subtype-Of	Repair delayed
Customer Complaint	Repair Problem	Types	Subtype-Of	Repair inadequate
Customer Complaint	Repair Problem	Types	Subtype-Of	Spare part not available
Customer Complaint	Repair Problem	Types	Subtype-Of	unauthorized repair
Customer Complaint	Sales Methods	Types	Subtype-Of	High pressure selling
Customer Complaint	Sales Methods	Types	Subtype-Of	Unsolicited commercial communication s
Customer Complaint	Sales Methods	Types	Subtype-Of	Unsolicited merchandise
Customer Complaint	Sales Methods	Types	Subtype-Of	Unsolicited service
Customer Complaint	Sales Office	located_in	is-of	Address
Customer Complaint	Sales promotion	Types	Subtype-Of	Hidden charges
Customer Complaint	Sales promotion	Types	Subtype-Of	Illegal lottery
Customer Complaint	Sales promotion	Types	Subtype-Of	Prize not received
Customer Complaint	Sales promotion	Types	Subtype-Of	Unfair contest
Customer Complaint	Sales promotion	Types	Subtype-Of	Unfair packaging

-D

Customer Complaint	Service Problem	Types	Subtype-Of	Service cancelled by provider
Customer Complaint	Service Problem	Types	Subtype-Of	Service inadequately per-formed
Customer Complaint	Service Problem	Types	Subtype-Of	Service not ordered
Customer Complaint	Service Problem	Types	Subtype-Of	Service not provided
Customer Complaint	Service Problem	Types	Subtype-Of	Service partially provided
Customer Complaint	Service Problem	Types	Subtype-Of	Service provision delayed
Customer Complaint	Symbolic Resolution	Types	Subtype-Of	Apologize
Customer Complaint	Symbolic Resolution	Types	Subtype-Of	Privacy Request
Customer Complaint	Third Party	Has	is-of	Address
Customer Complaint	Third Party	Has	is-of	Function
Customer Complaint	Third Party	Has	is-of	Third Party Name
Customer Complaint	Unexpected charge	Types	Subtype-Of	Delivery charge problem
Customer Complaint	Unexpected charge	Types	Subtype-Of	Incorrect amount
Customer Complaint	Unexpected charge	Types	Subtype-Of	Incorrect date
Customer Complaint	Unexpected charge	Types	Subtype-Of	Incorrect interest charge
Customer Complaint	Unexpected charge	Types	Subtype-Of	Price increase
Customer Complaint	Unexpected charge	Types	Subtype-Of	Supplementary (charge problem)
Customer	Unexpected	Types	Subtype-Of	Unjustified

-D

Complaint	charge			payment demand

-D

Appendix D: Thesis Glossary

Appendix D: Thesis Glossary

In this appendix, we present the definitions of some important terminology that we use in this thesis.

Axiomatization: an articulation or specification of knowledge (about a certain subject-matter) as a set of axioms.

Alternative axiomatization: are different formalizations of the same subject-matter.

Ontology rule: an axiom, a well-formed formulae in order to specify and constrain the legal models on an ontology. In conceptual data modeling, they are commonly called “constraints”. Notice that rules can be used for e.g. enforce integrity, derivation and inference, taxonomy, etc.

Conceptual relation: we use the terms ‘Conceptual relation’, ‘relation’, or ‘relationship’ to refer to n-ary relation. In this thesis, the term ‘concept’ commonly refers to a unary conceptual relation such as Person(Mustafa); also the term ‘relation’ is commonly used to refer to a binary or more conceptual relations such as WorksFor(Person, University).

Concept: a set of rules in our mind about a certain thing in reality.

Conceptualization: an intensional semantic structure, which encodes the implicit rules constraining the structure of a piece of reality [G98a].

Domain level: commonly accepted assumptions (i.e. understanding) about a piece of the reality. This term is often interchanged with the term “ontology level” to mean the same thing.

Epistemology level: The level that deals with the knowledge structuring primitives (e.g. concept types, structuring relations, etc.). [B79] [G94].

Extrinsic properties: “Extrinsic properties are not inherent, and they have a relational nature, like “being a friend of John”. Among these, there are some that are typically assigned by external agents or agencies, such as having a specific social security number, having a specific customer id., even having a specific name" [GW00].

Generic task: a highly reusable kind of task.

Intrinsic properties: "An intrinsic property is typically something inherent to an individual, not dependent on other individuals, such as having a heart or having a fingerprint" [GW00].

Extensional verses Intensional semantics: "The extensional semantics (value or denotation) of the expressions of a logic are relative to a particular interpretation, model, or situation. The extensional semantics of CarPool World, for example, are relative to a particular day. The denotation of a proposition is either True or False. If P is an expression of some logic, we will use [[P]] to mean the denotation of P. If we need to make explicit that we mean the denotation relative to situation S, we will use $[[P]]_S$. The intensional semantics (or intension) of the expressions of a logic are independent of any specific interpretation, model, or situation, but are dependent only on the domain being conceptualized. If P is an expression of some logic, we will use [P] to mean the intension of P. If we need to make explicit that we mean the intension relative to domain D, we will use $[P]_D$. Many formal people consider the intension of an expression to be a function from situations to denotations. For them, $[P]_D(S) = [[P]]_S$. However, less formally, the intensional semantics of a wfp can be given as a statement in a previously understood language (for example, English) that allows the extensional value to be determined in any specific situation." [S95].

Ontology reusability: the ability of using an ontology (or part of it) among several kinds of (autonomously specified) tasks.

Ontology usability: the ability of using an ontology among applications that perform the same kind of task.

State of affairs: A state of affairs refers to a particular instance of reality, or also called *a* possible world [WG03].

Bibliography

[A97a] Agnesund, M.: Representing culture-specific knowledge in a multilingual ontology. Proceedings of the IJCAI-97 Workshop on Ontologies and Multilingual NLP (1997)

[A97b] Appleton, B.: Patterns and Software: Essential Concepts and Terminology. Object Magazine Online. Vol. 3, No 5. May, (1997)

[ABA02] ABA Task Force on Electronic Commerce And Alternative Dispute Resolution. Final Report August, (2002)

[ACC04] Karl Aberer, Tiziana Catarci, Philippe Cudre-Mauroux, Tharam Dillon, Stephan Grimm, Mohand-Said Hacid, Arantza Illarramendi, Mustafa Jarrar, Vipul Kashyap, Massimo Mecella, Eduardo Mena, Erich Neuhold, Aris Ouksel, Thomas Risse, Monica Scannapieco, Felix Saltor, Luca De Santis, Stefano Spaccapietra, Steffen Staab, Rudi Studer, and Olga De Troyer: Emergent Semantics Systems. In M. Bouzeghoub, C. Goble, V. Kashyap, S. Spaccapietra, (eds): Proceedings of the first International IFIP Conference on Semantics of a Networked World. Volume 3226, LNCS, pages:14-44 Springer. ISBN: 3540-236090. Paris, France. June 2004.

[ACFOH03] Abdelali, A., Cowie, J., Farwell, D., Ogden, B., Helmreich, s.: Cross-Language Information Retrieval using Ontology. Proceedings of TALN Batz-sur-Mer. France. (2003)

[AKS04] Angelova,G., Kalaydjiev, O., Strupchanska, A.: Domain Ontology as a Resource Providing Adaptivity in eLearning. Proceedings of On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, LNCS 3292, Cyprus. (2004) pp. 700–712

-D

[AM04] Amir E., McIlraith s.: Partition-based logical reasoning for first-order and propositional theories. Artificial Intelligence Journal. (2004)

[AR00] Aitken, S., Reid, s.: Evaluation of an Ontology-Based Information Retrieval Tool. Proceedings of ECAI'00. Berlin, Germany. (2000)

[B01] Bryan, M. (eds.): MULECO -- Multilingual Upper-Level Electronic Commerce Ontology. MULECO draft CWA. At the CEN/ISSS Electronic Commerce Workshop (2001) http://xml.coverpages.org/Bryan-CWA-12-01.pdf (January 2004).

[B79] Brachman., R.: On the Epistemological Status of Semantic Networks," In: Findler,N. (ed.). Associative Networks: Representation and Use of Knowledge by Computers. Academic Press, New York. (1979)

[BB03] Borgida, A., Brachman, R.: Conceptual Modeling with Description Logics. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., (eds.): The Description Logic Handbook, Theory, Implementation and Applications. ISBN: 0521781760 (2003)

[BB04] Beneventano, D., Bergamaschi, S.: The MOMIS methodology for Integrating Heterogeneous Data Sources. IFIP World Computer Congress. Toulouse, France. August (2004)

[BBB+98] Baker, G., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. In: Glasgow, J., Littlejohn, T., Major, F., Lathrop, R., Sankoff D., Sensen, S. (eds.): 6th Int. Conf. on Intelligent Systems for Molecular Biology. AAAI Press, Menlo Park. Montreal, Canada. (1998) pp 25–34

[BBDD97] Briand, L.C., Bunse, C., Daly, J.W. and Differding, C.: An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents. In: Empirical Software Engineering, Vol. 2, No. 3. (1997) pp. 291–312.

[BBH96] Beys, P., Benjamins, R., van Heijst, G.: Remedying the reusability-usability trade-off for problem solving methods. In: B.R. Gaines and M. Mussen, (eds.): Proceedings of the KAW'96. Banff, Ca, (1996)

[BC88] Bylander, T., Chandrasekaran, B.: Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In: Gaines B., Boose, J. (eds.): Knowledge Acquisition for Knowledge Based Systems. Vol. 1. Academic Press, London. (1988) pp. 65–77

[BCD01] Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML Class Diagrams using Description Logic Based Systems. Workshop on Applications of Description Logics - ADL'01. (2001)

[BCFF04] Bonino, D., Corno, F., Farinetti, L., Ferrato, A.: Multilingual Semantic Elaboration in the DOSE platform. ACM Symposium on Applied Computing, SAC'04. Nicosia, Cyprus. March (2004)

[BCMNP03] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press. (2003)

[BCW02] Brewster, C., Ciravegna, F., Wilks, Y.: User-Centred Ontology Learning for Knowledge Management. Proceedings of the 7th FInternational Conference on Applications of Natural Language to Information Systems, Stockholm, Lecture Notes in Computer Science 2553, Springer Verlag. June (2002)

[BCW97] Barley, M., Clark, P., Williamson, K., Woods, S.: The neutral representation project. Proceedings of AAAI'97 Spring Symposium on Ontological Engineering. AAAI Press. (1997)

[BDMW95] Birmingham, W., Durfee, E., Mullen, T., Wellman, M.: The Distributed Agent Architecture Of The University of Michigan Digital Library (UMDL). AAAI Spring Symposium Series on Software Agents. (1995)

[BDVHP00] Brejova, B., DiMarco, C., Vinar, T., Hidalgo, S. R., Holguin, G. and Patten, C. Finding Patterns in Biological Sequences. Unpublished project report for CS798G, University of Waterloo, Fall 2000.

[BF99] Berners-Lee, T., Fischetti, M.: Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor. Harper, San Francisco. (1999)

[BH96] Bloesch, A., Halpin, T.: ConQuer: a Conceptual Query Language. In: Thalheim, B. (ed.): Proceedings of Conceptual Modeling - ER'96. Lecture Notes in Compute Science, Springer-Verlag. (1996) pp. 121–33

[BHGSS03] Bouquet, P., van Harmelen, F., Giunchiglia, F., Serafini, L., Stuckenschmidt H.: C-OWL: Contextualizing ontologies. Proceedings of the second International Semantic Web Conference - ISWC'03, Sanibel Island, Florida. October (2003)

[BHW91] van Bommel, P., ter Hofstede, A.H.M. , van der Weide, Th.P. : Semantics and verification of object role models. Information Systems, 16(5). October (1991) 471–495

[BLA+05] de Bruijn, J., Lara, R., Arroyo, S., Gomez, J., Han, S., Fensel, D.: A Unified Semantic Web Services Architecture based on WSMF and UPML. The International Journal of Web Engineering and Technology (IJWET). (2005)

[BM99] Bench-Capon T.J.M., Malcolm G.: Formalising Ontologies and Their Relations. Proceedings of DEXA'99. (1999) pp. 250–259

[BS03] Borgida A., Serafini L.: Distributed Description Logics: Assimilating Information from Peer Sources. In: Aberer K., March S., and Spaccapietra S., (eds.): Journal on Data Semantics, Vol. 2800. LNCS, Springer, ISBN: 3-540-20407-5. October (2003) pp. 153–184

[BVW97] Breuker, J., Valente, A., Winkels, R.: Legal ontologies: a functional view. In: Visser, P., Winkels, R. (eds.): Legal Ontologies. ACM, New York. (1997) pp. 23–36

[C92] Clancey W.J., "Model construction operators". Artificial Intelligence, 53(1):1-115, (1992).

[C98] Chalabi, C.: Sakhr Arabic-English Computer-Aided Translation System. AMTA'98. (1998) pp. 518–521

[CAA06] Philippe Cudre-Mauroux, Karl Aberer, Alia Abdelmoty, Tiziana Catarci, Ernesto Damiani, Arantxa Illaramendi, Mustafa Jarrar, Robert Meersman, Erich Neuhold, Christine Parent, Kai-Uwe Sattler, Monica Scannapieco, Stefano Spaccapietra, Peter Spyns, and Guy De Tre: Viewpoints on Emergent Semantics. In Stefano Spaccapietra, Karl Aberer, Philippe Cudre-Mauroux (eds): Journal on Data Semantics. 4090(6):1-27. ISBN: 3540367128. Springer. 2006.

[CBB+04] Collet, C., Belhajjame, K., Bernot, G., Bobineau, C., Bruno, G., Finance, B., Jouanot, F., Kedad, Z., Laurent, D., Tahi, F., Vargas-Solar, G., Tuyet-Trinh V.: Towards a mediation system framework for transparent access to largely distributed sources. Proceedings of the International Conference ICSNW. (2004)

[CC03] Chia-Wei, W., Chao-Lin, L.: Ontology-based Text Summarization for Business News Articles. Computers and Their Applications. (2003) pp. 389–392

[CDLNR98] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Information integration: Conceptual modeling and reasoning support. In Proceedings Of the 6th International Conference on Cooperative Information Systems (CoopIS'98). (1998) pp. 280-291

[CG01] Corcho, O., Gmez-Prez, A.: Solving Integration Problems of E-Commerce Standards and Initiatives through Ontological Mappings. Proceedings of IJCAI01. (2001)

[CHP01] Cranefield, S., Haustein, S., Purvis, M.: UML-Based Ontology Modelling for Software Agents. Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents. Montreal. (2001) pp. 21–28

[CIHF02] Cho, Y., Im, I., Hiltz, S., Fjermestad, J.: An Analysis of Online Customer Complaints: Implications for Web Complaint Management. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS’02). Volume 7. Hawaii. (2002)

[CJ93] Chandrasekaran, B., Johnson, T.: Generic Tasks and Task Structures: History, Critique and New Directions. In: David, J., Krivine, J., Simmons, R. (eds.): Second Generation Expert Systems, Springer. (1993) pp. 233–272

[CJB99] Chandrasekaran, B., Johnson, R., Benjamins, R.: Ontologies: what are they? why do we need, them?. IEEE Intelligent Systems and Their Applications. 14(1). Special Issue on Ontologies. (1999) pp. 20–26.

[CP99] Cranefield, S., Purvis, M.: UML as an ontology modelling language. Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence, IJCAI’99, (1999)

[CW87] Claes, F., Wernerfelt, B.: Defensive Marketing Strategy by Customer Complaint Management: A Theoretical Analysis. Journal of Marketing Research, No. 24. November (1987) pp. 337–346

[DF01] Ding, Y., Fensel, D.: Ontology library systems: the key for successful ontology reuse. Proceedings of the first Semantic Web Working Symposium, Stanford, CA, USA. August (2001)

[DFJ05] Tharam Dillon, Ling Feng, Mustafa Jarrar, Aldo Gangemi, Joost Breuker, Jos Lehmann, and Andre Valente (eds): Proceedings of the 1st IFIP WG 2.12 and WG 12.4 International Workshop on Web Semantics (SWWS'06). In OTM Workshops. Volume 3762 of LNCS. Springer. ISBN: 3540297391. Larnaca, Cyprus. November 2005.

[DHHS01] Degen, W., Heller, B., Herre, H. and Smith, B.: GOL: Towards an Axiomatized Upper-Level Ontology. In: Welty, C., Smith B. (eds.): Formal Ontology in Information Systems. Proceedings of the Second International Conference (FOIS 2001). ACM Press. New York: October (2001) pp. 34–46

[DJM02a]: Demey, J., Jarrar, M., Meersman, R.: A Conceptual Markup Language that supports interoperability between Business Rule modeling systems. Proceedings of the Tenth International Conference on Cooperative Information Systems (CoopIS 02). Springer Verlag LNCS 2519. (2002) pp. 19–35

[DJM02b]: Demey, J., Jarrar, M., Meersman, R.: Markup Language for ORM Business Rules. In: Schroeder M. & Wagner G. (eds.), Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (RuleML’02). (2002) pp. 107–128

[DMV] De Troyer, O., Meersman, R., Verlinden, P.: RIDL* on the CRIS case: A Workbench for NIAM. Technical report. INFOLAB, Tilburg University, The Netherlands.

[DW00] Deridder, D., Wouters, B.: The Use of an Ontology to Support a Coupling between Software Models and Implementation. European

Conference on Object-Oriented Programming (ECOOP'00), International Workshop on Model Engineering. (2000)

[E05] Embley, D.: Toward Tomorrow's Semantic Web -- An Approach Based on Information Extraction Ontologies. Position Paper for Dagstuhl Seminar. January (2005)

[EN99] Elmasri, R., Navathe, S.: Fundamentals of Database Systems. (3rd Edition). Addison-Wesley Publishing. (1999)

[EWHLF02] Elmasri, R., Wu, Y., Hojabri, B., Li, C., Fu, J.: Conceptual Modeling for Customized XML Schemas. In: Spaccapietra, S., March, s., Kambayashi, Y. (Eds.): Proceedings of 21st International Conference on Conceptual Modeling (ER'02). Tampere, Finland. Lecture Notes in Computer Science 2503 Springer. ISBN 3-540-44277-4. October (2002) pp. 429–443

[F02] Franconi, E.: Tutorial on Description Logics for Conceptual Design, Information Access, and Ontology Integration: Research Trends. Proceedings of the 1st International Semantic Web Conference (2002)

[F97] Frank, A.: Spatial Ontology: A Geographical Point of View. In: Stock, O. (eds.): Spatial and Temporal Reasoning., Kluwer Academic Publishers, Dordrecht, The Netherlands. (1997) pp. 135–153

[FE99] Fonseca, F., Egenhofer, M.: Ontology-Driven Geographic Information Systems. In: the 7th ACM Symposium on Advances in Geographic Information Systems. Kansas City, MO: ACM Press, N.Y. (1999)

[FGJ97] Fernandez, M., Gomez-Perez, A., Juristo, N.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Workshop on Ontological Engineering. Spring Symposium Series. AAAI97 Stanford, USA. (1997)

[FLS96] Falasconi, S., Lanzola, G., Stefanelli. M.: Usingontologies in multi-agent systems. In: Proceedings of Tenth Knowledge Acquisition for Knowledge-BasedSystems Workshop (KAW'96). (1996)

[G02] Guarino, N.: Ontology-Driven Conceptual Modelling. Tutorial at 21st International Conference on Conceptual Modeling (ER'02). Tampere, Finland.. (2002)

[G04] Gangemi, A.: Some design patterns for domain ontology building and analysis. An online presentation at (http://www.loa-cnr.it/Tutorials/OntologyDesignPatterns.zip April 2004)

[G85] Gilberg, R.:A Schema methodology for Large Entity-Relationship Diagrams. Proceedings of the 4th International Conference on Entity-Relationship Approach. Chicago, Illinois. ISBN O-13186-0645-2. October (1985) pp. 320–327

[G94] Guarino, N.: The Ontological Level. In R. Casati, B. Smith and G. White (eds.), Philosophy and the Cognitive Science. Hölder-Pichler-Tempsky, Vienna: 443-456. (1994)

[G95] Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43(5/6) (1995)

[G97] Guarino, N.: Understanding, building, and using ontologies: A commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga." International Journal of Human and Computer Studies No. 46. (1997) pp. 293–310

[G98a] Guarino, N.: Formal Ontology in Information Systems. Proceedings of FOIS'98, IOS Press, Amsterdam. (1998) pp. 3–15

[G98b] Guarino, N.: Some Ontological Principles for Designing Upper Level Lexical Resources. In: A. Rubio, N. Gallardo, R. Castro and A. Tejada (eds.): Proceedings of First International Conference on

Language Resources and Evaluation. ELRA - European Language Resources Association, Granada, Spain. (1998)

[GAC+04] Glaser, H., Alani, H., Carr, L., Chapman, S., Ciravegna, F., Dingli, A., Gibbins, N., Harris, S., schraefel, m. c. Shadbolt, N.: CS AKTive Space: Building a Semantic Web Application. In: Bussler, C., Davies, J., Fensel, D. and Studer, R. (Eds.): First European Web Symposium (ESWS'04). Springer Verlag. (2004) pp. 417–432

[GB99] Gomez-Perez, A., Benjamins, R.: Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. Proceedings of the IJCAI-99, Workshop on Ontologies and Problem-Solving Methods (KRR5), MorganKaufmann (1999)

[Gene00] Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium Nature Genet. No. 25. (2000) pp. 25–29

[GG01] Giunchiglia, F., Ghidini, C.: Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence journal, 127(2). (2001) pp. 221–259

[GG95] Guarino, N. and Giaretta, P., "Ontologies and Knowledge Bases: Towards a Terminological Clarification"` in: Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, N. Mars (ed.), pp 25-32, IOS Press, Amsterdam (1995).

[GG95] Guarino, N. and Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. In: Mars, N. (eds.): Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press. Amsterdam (1995) pp. 25–32

[GGMO01] Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A.: Understanding toplevel ontological distinctions. Proceedings of IJCAI-01 Workshop on Ontologies and Information Sharing. AAAI Press. Seattle, USA, (2001) pp. 26–33

-D

[GGO02] Gangemi, A., Guarino, N., Oltramari A., Borgo, S.: Cleaning-up WordNet's top-level. Proceedings of the 1st International WordNet Conference. January (2002)

[GGV97] Gilarranz, J., Gonzalo, J., Verdejo, F.: Language-independent text retrieval with the EuroWordNet multilingual semantic database. The Second Workshop on Multilinguality in the Software Industry: The AI Contribution. August (1997)

[GHW02] Guizzardi, G., Herre, H., Wagner G.: Towards Ontological Foundations for UML Conceptual Models. proceedings of the 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), Lecture Notes in Computer Science, Vol. 2519, Springer-Verlag, Berlin. (2002) pp. 1100–1117

[GMV99] Guarino, N., Masolo, C., Vetere, G.: OntoSeek: Content-Based Access to the Web. IEEE Intelligent Systems. June (1999) pp. 70–80.

[GN87] Genesereth, M.R., Nilsson, N.J.: Logical Foundation of Artificial Intelligence. Morgan Kaufmann. Los Altos, California. (1987)

[GP01] Gangemi A., Pisanelli DM., Steve G.: A formal Ontology Framework to represent Norm Dynamics. Proceedings of Second International Workshop on Legal Ontologies, Amsterdam, NL. (2001)

[GP03] Guarino, N., Persidis A.: Evaluation Framework for Content Standards. Deliverable 3.5, OntoWeb EU project (IST-2000-29243), (2003)

[GPB99] Gomez-Perez, A., Benjamins, R.: Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods. Morgan-Kaufmann (1999)

[GW00] Guarino, N., Welty, C.: A Formal Ontology of Properties. Proceedings of the ECAI-00 Workshop on Applications of Ontologies and Problem Solving Method. Berlin, Germany. (2000) pp. 12.1–12.8

-D

[GW02] Guarino, N. and Welty, C.: Evaluating Ontological Decisions with OntoClean. Communications of the ACM, 45(2). (2002) pp. 61–65

[H01] Halpin, T.: Information Modeling and Relational Databases. 3rd edn. Morgan-Kaufmann. (2001)

[H89] Halpin, T.: A logical analysis of information systems: static aspects of the data-oriented perspective. PhD thesis, University of Queensland, Brisbane. Australia. (1989)

[H97] Halpin, T.: An Interview- Modeling for Data and Business Rules. In: Ross, R. (eds.): Database Newsletter. vol. 25, no. 5. (Sep/Oct 1997). -This newsletter has since been renamed Business Rules Journal and is published by Business Rules Solutions, Inc.

[H99] Halpin, T.: UML data models from an ORM perspective: Part 7. Journal of Conceptual Modeling. InConcept. February (1999)

[Hj01] Heflin, J.: Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. Ph.D. Thesis, University of Maryland, College Park. (2001)

[HP95] Halpin, T., Proper, H.: Subtyping and polymorphism in object-role modeling. Data & Knowledge Engineering 15(3). (1995) pp. 251–281

[HPW93] ter Hofstede, A., Proper, H., van der Weide, T.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18(7). October (1993) pp. 471–495

[HS01] Horrocks I., Sattler, U.: Ontology reasoning in the SHOQ(D) description logic. In: Nebel, B. (eds.): Proceedings of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI'01). Morgan Kaufmann. (2001) pp. 199–204

-D

[HST99] Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.): Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99). Lecture Notes in Artificial Intelligence 1705, Springer-Verlag. (1999) pp. 161–180.

[HSW97] van Heijst, G., Schreiber, A., Wielinga, B.: Using Explicit Ontologies in KBS Development. International Journal of Human-Computer Studies, 46. (1997) pp. 183–292

[HV93] Hemmann, T., Voss, H.: A Reusable and Specializable Interpretation Model for ModelBased Diagnosis. In: Luckenhoff, C., Fensel, D., Studer, D. (eds.): Proceeding 3rd KADS Meeting Siemens AG. Munich. March (1993) pp. 189–205

[Inn+03] Persidis A., Niederée C., Muscogiuri C., Bouquet P., Wynants M.: Innovation Engineering for the Support of Scientific Discovery. Innovanet Project (IST-2001-38422), deliverable D1. (2003)

[J05a] Mustafa Jarrar: Modularization and automatic composition of Object-Role Modeling (ORM) Schemes. In OTM 2005 Workshops, proceedings of the International Workshop on Object-Role Modeling (ORM'05). Volume 3762, LNCS, Pages (613-625), Springer. ISBN: 3540297391. November 2005.

[J06] Mustafa Jarrar: Towards the notion of gloss, and the adoption of linguistic resources in formal ontology engineering. In proceedings of the 15th International World Wide Web Conference (WWW2006). Edinburgh, Scotland. Pages 497-503. ACM Press. ISBN: 1595933239. May 2006.

[J07] Mustafa Jarrar: Towards Automated Reasoning on ORM Schemes. -Mapping ORM into the DLR_idf description logic. Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007).

Volume 4801, LNCS, Pages (181-197), Springer. ISBN:9783540755623. New Zealand. November 2007.

[J07a] Mustafa Jarrar: ORM Markup Language, version 3. Technical Report. STAR Lab, Vrije Universiteit Brussel, Belgium. January 2007

[J07b] Mustafa Jarrar: Mapping ORM into the SHOIN/OWL Description Logic- Towards a Methodological and Expressive Graphical Notation for Ontology Engineering. In OTM workshops, proceeding of the International Workshop on Object-Role Modeling (ORM'07). Volume 4805, LNCS, Pages (729-741), Springer. ISBN: 9783540768890. Portogal. November, 2007

[J08] Mustafa Jarrar: Towards Effectiveness and Transparency in e-Business Transactions, An Ontology for Customer Complaint Management. A book chapter in "Semantic Web Methodologies for E-Business Applications". Idea Group Inc. (2008) (To appear)

[JCCP06] Mustafa Jarrar, Claude Ostyn, Werner Ceusters, and Andreas Persidis (eds): Proceedings of the International Workshop on Ontology content and evaluation (OnToContent 2006). In OTM Workshops (2). Volume 4278 of LNCS. page (1011), Springer Berlin. ISBN: 9783540482734. Montpellier, France. November 2006.

[JD06] Mustafa Jarrar and Mohammed Eldammagh: Reasoning on ORM using Racer. Technical Report. STAR Lab, Vrije Universiteit Brussel, Belgium. August 2007

[JDM03] Jarrar M., Demy J., Meersman R.: On Using Conceptual Data Modeling for Ontology Engineering. In: Aberer K., March S., and Spaccapietra S., (eds.): Journal on Data Semantics, Special issue on "Best papers from the ER/ODBASE/COOPIS 2002 Conferences", LNCS Vol. 2800, Springer. ISBN: 3-540-20407-5. October (2003) pp. 185–207

[JH08] Mustafa Jarrar and Stijn Heymans: Towards Pattern-based Reasoning for Friendly Ontology Debugging. Journal of artificial tools. 2008. (To appear).

[JKD06] Mustafa Jarrar, Maria Keet, and Paolo Dongilli: Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report. STARLab, Vrije Universiteit Brussel, February 2006.

[JLVM03] Jarrar, M., Lisovoy, A., Verlinden, R., Meersman, R.: "Ontoform" Ontology based CCForms demo. Deliverable 6.3, CCForm Project (IST-2001-34908), 5th framework. Brussels (2003)

[JM02a] Jarrar, M., Meersman, R.: Formal Ontology Engineering in the DOGMA Approach. In: 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02). Lecture Notes in Computer Science, Vol. 2519, Springer-Verlag. Berlin (2002) pp. 1238–1254

[JM02b] Jarrar, M., Meersman, R.: Scalability and Knowledge Reusability in Ontology Modeling. Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine (SSGRR'2002s) (2002)

[JM08] Mustafa Jarrar and Robert Meersman: Ontology Engineering -The DOGMA Approach. In Elizabeth Chang and Tharam Dillon and Robert Meersman and Katia Sycara (eds): Advances in Web Semantic. Volume 1, A state-of-the Art Semantic Web Advances in Web Semantics IFIP2.12. Chapter 3. Springer. 2008.

[JS03] Jarrar, M., Salaun, A. (eds.): Proceedings: Regulatory ontologies and the modeling of complaint regulations (WORM CoRe 2003). Workshop held at the "On the Move to Meaningful Internet Systems 2003" conference (OTM'03). Catania, Sicily, Italy. Springer LNCS. November (2003)

-D

[JS06] Mustafa Jarrar and Stijn Heymans: Unsatisfiability Reasoning in ORM Conceptual Schemes. In Torsten Grust et al. (eds): Proceeding of International Conference on Semantics of a Networked World. Volume 4254, LNCS, Pages (517-534), Springer. ISBN: 3540467882. Munich, Germany, March 2006.

[JSOW07] Mustafa Jarrar, Andreas Schmidt, Claude Ostyn, and Werner Ceusters (eds): Proceedings of the International Workshop on Ontology content and evaluation (OnToContent 2007). In OTM Workshops (1). Volume 4805 of LNCS. page (509), Springer Berlin. ISBN: 978-3540768876. Algarve, Portugal. November 2007.

[JVM03] Jarrar, M., Verlinden, R., Meersman, R.: Ontology-based Customer Complaint Management. In: Jarrar M., Salaun A., (eds.): Proceedings of the workshop on regulatory ontologies and the modeling of complaint regulations, Catania, Sicily, Italy. Springer Verlag LNCS. Vol. 2889. November (2003) pp. 594–606

[K03] Khosla, R.: Multi-Layered Distributed Agent Ontology for Soft Computing Systems. Proceedings of the 17th International Conference on Knowledge-based Intelligent Information and Engineering Systems. Oxford, U.K. September (2003) pp. 445–52

[K04] Keet, M.: Aspects of ontology integration. Technical report. School of Computing, Napier University. January (2004)

[K96] Mahesh, K.: Ontology development for machine translation: Ideology and Methodology. Technical report MCCS-96-292. Memoranda in Computer and Cognitive Science. New Mexico State University, Computing Research Laboratory, Las Cruces, NM. (1996)

[KF01] Klein, M. and Fensel, D.: Ontology Versioning on the Semantic Web. The First International Semantic Web Working Symposium (SWWS'01) (2001)

-D

[KKOF02] Klein, M., Kiryakov, A., Ognyanov, D., Fensel, D.: Ontology versioning and change detection on the web. The 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02). Sig uenza, Spain. October (2002)

[KN03] Klein, M., Noy.: A component-based framework for ontology evolution. Technical Report IR-504, Department of Computer Science, Vrije Universiteit Amsterdam. March (2003)

[KRS+02] Karp, P.D., Riley, M., Saier, M., Paulsen, I.T., Paley, S., Pellegrini-Toole, A.: The Ecocyc Database. Nucleic Acids Research, 30(1):56. (2002)

[KTT03] Kerremans, K., Temmerman, R. and Tummers, J.: Representing multilingual and culture-specific knowledge in a VAT regulatory ontology: support from the termontography approach. In: Meersman, R., Tari, Z. (eds.) OTM 2003 Workshops. Tübingen: Springer Verlag. (2003)

[LWP+02] Lauser, B., Wildemann, T., Poulos, A., Fisseha, F., Keizer, J., Katz, S.: A Comprehensive Framework for Building Multilingual Domain Ontologies: Creating a Prototype Biosecurity Ontology. Proceedings of the International Conference on Dublin Core and Metadata for e-Communities. Firenze University Press. Florence, Italy October (2002) pp. 113–123

[M00] Meersman R.: Can Ontology Theory Learn from Database Semantics?. Proceedings of the Dagstuhl Seminar 0121 'Semantics on the Web' (2000)

[M01a] Meersman, R.: Ontologies and Databases: More than a Fleeting Resemblance. In: d'Atri A., Missikoff, M. (eds.): OES/SEO 2001 Rome Workshop, Luiss Publications (2001)

[M01b] Meersman R.: New Frontiers in Modeling Technology: The Promise of Ontologies. Proceedings of the SISO ESM Conference on Simulation (2001)

[M04] Mika, P.: Social Networks and the Semantic Web. IEEE/WIC/ACM International Conference on Web Intelligence (WI'04). IEEE Computer Society. ISBN 0-7695-2100-2. Beijing, China. (2004) pp. 285–291

[M55] Martinet, A.: Economie des changements phonétiques, Berne: Francke, (1955) pp. 157-158

[M81] Meersman, R.: Languages for the High-Level End User. InfoTech State of the Art Report. Pergamon Press. (1981)

[M86] Meersman, R.: Knowledge and Data: A Survey in the Margin of the IFIP DS-2 Conference. In: Spacapietra, S., (eds.): Entity-Relationship Approach: Ten Years of Experience in Information Modeling, Proceedings of the Fifth International Conference on Entity-Relationship Approach. North-Holland. Dijon, France (1986) pp. 25–34

[M93] McCarthy, J.: Notes on Formalizing Context. Proceedings of IJCAI'93. Morgan-Kaufmann. (1993)

[M95] Meersman, R.: An essay on the Role and Evolution of Data(base) Semantics. In: Meersman, R., Mark L. (eds.): Proceeding of the IFIP WG 2.6 Working Conference on Database Applications Semantics (DS-6). CHAPMAN & HALL. Atlanta, USA. (1995)

[M98] Musen, M.: Domain Ontologies in Software Engineering: Use of Protege with the EON Architecture. Methods of Information in Medicine, No. 37. (1998) pp. 540–550

[M99a] Meersman R.: The Use of Lexicons and Other Computer-Linguistic Tools. In: Zhang Y., Rusinkiewicz M, & Kambayashi Y. (eds.): Semantics, Design and Cooperation of Database Systems, The

International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'99). Springer Verlag. Heidelberg. (1999) pp. 1–14

[M99b] Meersman R., Semantic Ontology Tools in Information System Design. In, Ras, Z. & Zemankova, M.,(eds.), Proceedings of the ISMIS 99 Conference, LNCS 1609, Springer Verlag. (1999) pp. 30–45

[MBFGM90] Miller, G. Beckwith, R., Fellbaum, F., Gross, D., Miller, K.: Introduction to wordnet: an on-line lexical database. International Journal of Lexicography, 3(4). (1990) pp. 235–244

[MBGGO03] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: WonderWeb Deliverable D18, Ontology Library. IST Project 2001-33052 WonderWeb, Deliverable D18. (2003)

[MC02] Meisel, H., Compatangelo, E.: EER-ConcepTool: a "reasonable" environment for schema and ontology sharing. Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02), IEEE Computer Society Press. (2002) pp. 527–534

[MMS03] Maedche, A., Motik B., Stojanovic L.: Managing multiple and distributed ontologies on the Semantic Web. The International Journal on Very Large Data Bases. Springer-Verlag New York, Vol. 12, number 4, issn: 1066-8888. (2003) pp. 286–302

[MVBCFGG04] Masolo, C. Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social Roles and their Descriptions. Proceeding of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR'04). Canada. (2004)

[N90] Nöth, W.: Handbook of Semiotics. Bloomington, IN : Indiana University Press (1990)

[N94] Nonaka, I: A dynamic theory of organizational knowledge creation. In: Organizational Science, Vol. 5, No. 1. (1994) pp. 14–37

[NCMSC00] da Nóbrega, M., Castro, E., Malbos, P., Sallantin, J., Cerri, A.: A framework for supervised conceptualizing. In: Benjamins, V. R., Gómez Pérez, A., Guarino, N., Uschold, M. (eds.): Workshop on Applications of Ontologies and Problem-Solving Methods (ECAI–00). Berlin, Germany. (2000)

[NM02] Nakhimovsky, A., Myers, T.: Web Services: Description, Interfaces and Ontology. In: Geroimenko, V., Chen, C. (eds.): Visualizing the Semantic Web. Springer. ISBN 1-85233-576-9. (2002) pp. 135–150.

[P04] Pan, J.: Description Logics: Reasoning Support for the Semantic Web. Ph.D. Thesis, School of Computer Science, the University of Manchester. (2004)

[P05] Pretorius, A. J.: Visual Analysis for Ontology Engineering. Journal of Visual Languages and Computing. (2005)

[P72] Parnas, D. L.: On the criteria to be used in decomposing system into modules. Communications of the ACM, Vol. 15, No. 12. December (1972) pp. 1053–1058

[P96] Polany, M.: The Tacit Dimension. Doubleday, Garden City-N.Y. (1996)

[PFP+92] Patil, R., Fikes, R., Patel-Schneider, P., McKay, D., Finin, T., Gruber, T., Neches, R.: The DARPA Knowledge Sharing Effort: Progress Report. Proceedings of Knowledge Representation and Reasoning. (1992) pp. 777–788

[PSDM03] Reinberger M.-L., Spyns P., Daelemans W. Meersman R.: Mining for lexons: applying unsupervised learning methods to create ontology bases. In: Meersman R., Zahir T., Schmidt D. et al.,(eds.), On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, LNCS 2888, Springer Verlag. (2003) pp. 803–819

[Q91] Qmair, Y.: Foundations of Arabic philosophy. Dar al-Shoroq. Bairut, ISBN 2-7214-8024-3. (1991)

[R00] Richards. D. "The Reuse of Knowledge: A User-Centered Approach", International Journal of Human Computer Studies, (2000).

[R03] Rector, A.: Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. Proceedings of the international conference on Knowledge captureIsland. ACM Press. ISBN:1-58113-583-1. FL, USA. (2003) pp. 121–128

[R70] Royce, W.: Managing the development of large software systems: Concepts and techniques. Proceedings of WESCON. August (1970)

[R88] Reiter, R.: Towards a Logical Reconstruction of Relational Database Theory. In: Mylopoulos, J., Brodie, M.L. (eds.): Readings in AI and Databases. Morgan Kaufman. (1988)

[RFOGP99] Rigoutsos, I., Floratos, A., Ouzounis, C., Gao, Y. & Parida, L.: Dictionary building via unsupervised hierarchical motif discovery in the sequence space of natural proteins. Proteins: Struct. Funct. Genet. 37. (1999) pp. 264–277.

[RS93] Rauh, O., Stickel, E.: Searching for Compositions in ER Schemes. In: Elmasri R., Kouramajian, V. (eds.): Proceeding of the 12th Int. Conference on Entity Relationship Approach. Arlington, Texas. December (1993) pp. 75–86

[RSV98] Roberto, C., Smith, B., Varzi A.: Ontological tools for geographic representation. In: N. Guarino (eds.): Formal Ontology in Information Systems, Proceedings of the First International Conference (FOIS'98). Amsterdam IOS Press. Trento, Italy. June (1998) pp. 77–85

[RVMS99] Russ, T., Valente, A., MacGregor, R., Swartout, W.: Practical Experiences in Trading Off Ontology Usability and Reusability.

Proceedings of the Twelfth Banff Knowledge Acquisition for Knowledge-based Systems Workshop. (1999) pp. 4.11.1–4.11.20

[S00] Sowa, J.F.: Ontology, metadata, and semiotics. In: Ganter, B., Mineau, G.W., (eds.): Conceptual structures: logical, linguistic and computational issues: 8th international conference on conceptual structures (ICCS'00). Darmstadt Germany. Lecture Notes in Artificial Intelligence, 1867, Springer-Verlag Berlin. August (2000) pp. 55–81

[S02] Smith B.: Ontology and information systems. Stanford Encyclopedia of Philosophy (2002) http://ontology.buffalo.edu/ontology(PIC).pdf (January 2005)

[S03a] Smith, B.: Ontology. In: Floridi, L. (eds.): Blackwell Guide to the Philosophy of Computing and Information. Oxford: Blackwell. (2003) pp. 155–166

[S03b] Sure, Y.: Methodology, Tools and Case Studies for Ontology based Knowledge Management. PhD Thesis, University of Karlsruhe, Department of Economics and Business Engineering. (2003)

[S85] Shoval, P.: Essential information structure diagrams and database schema design. Information Systems, 10(4). (1985) pp. 417-423

[S93] Steels, L., "The componential framework and its role in reusability", in: Second Generation Expert Systems, J.-M. David, J.-P. Krivine & R.Simmons, (eds.), pp. 273-298. Berlin: Springer-Verlag (1993).

[S95] Shapiro, S.: Propositional, First-Order And Higher-Order Logics: Basic Definitions, Rules of Inference, Examples. In: Iwanska, L., Stuart, S., Shapiro, (eds.): Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language. AAAI Press/The MIT Press, Menlo Park, CA. (1995)

[S96] Siirtola: Managing Large Entity-Relationship Diagrams. In: Thalheim, B., Yigitbasi, S., (eds.): Proceeding of the Workshop ER CASE Tools. Cottbus, Germany. October (1996) pp. 29–42

[SCD06] Katia Sycara, Elizabeth Chang, Ernesto Damiani, Mustafa Jarrar, and Tharam Dillon (eds): Proceedings of the 2nd IFIP WG 2.12 and WG 12.4 International Workshop on Web Semantics (SWWS'06). In OTM Workshops (2). Volume 4278 of LNCS. page (1723), Springer Berlin. ISBN: 9783540482734. Montpellier, France. November 2006.

[SGG+05] Suárez-Figueroa, M., García-Castro, R., Gómez-Pérez, A., Palma R., Nixon, L., Paslaru, L., Hartmann J., Jarrar, J.: Identification of standards on metadata for ontologies. Deliverable D1.3.2. EU-IST Network of Excellence (NoE) IST-2004-507482 (KWEB), Luxemburg (2005)

[SGP98] Steve G., Gangemi A., Pisanelli D.M.: Integrating Medical Terminologies with the ONIONS Methodology. In: Kangassalo, H., Charrel, J.P. (eds.): Information Modeling and Knowledge Bases VIII. Amsterdam IOS Press (1998)

[SH05] Stuckenschmidt, H., van Harmelen, F.: Information sharing on the semantic Web. Springer. Berlin. ISBN 3-540-20594-2 (2005)

[SK03] Stuckenschmidt H., Klein M.: Modularization of Ontologies - WonderWeb: Ontology Infrastructure for the Semantic Web. Deliverable 21. WonderWeb Project (IST 2001-33052) (2003)

[SKC02] Sampson, D., Karagiannidis C., Cardinali, F.: An Architecture for Web-Based e-Learning Promoting Re-usable Adaptive Educational e-Content. Educational Technology & Society Journal of International Forum of Educational Technology & Society and IEEE Computer Society Learning Technology Task Force, ISSN 1436-4522, Special Issue on Innovations in Learning Technologies, 5(4), August (2002)

[SKKM03] Sunagawa, E., Kozaki, K., Kitamura, Y., and Mizoguchi R.: An Environment for Distributed Ontology Development Based on Dependency Management. Proceedings of the Second International SemanticWeb Conference (ISWC'03). Springer-Verlag, LNCS 2870. FL, USA. ISBN: 3-540-20362-1. (2003) pp. 453–468

[SM93] Swartout, W.R. and Moore, J.D., ”Explanation in Second Generation Expert Systems”, in: Second Generation Expert Systems, J.-M. David, J.-P. Krivine and R. Simmons (eds),. Berlin: Springer-Verlag (1993).

[SMD00] Shum, S., Motta, E., Domingue, J.: ScholOnto: an ontology-based digital library server for research documents and discourse. Int. J. on Digital Libraries 3(3). (2000) pp. 237-248

[SOV+02] Spyns, P., Oberle, D., Volz, R., Zheng, J., Jarrar, M., Sure, Y., Studer, R., Meersman, R.: OntoWeb - a Semantic Web Community Portal. In: Karagiannis, D., Reimer, U., (eds.): Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM'02), LNAI 2569, Springer Verlag. (2002) pp. 189–200

[SP94] Spaccapietra, S., Parent, C.: View Integration: A Step Forward in Solving Structural Conflicts. IEEE Transactions on Data and Knowledge Engineering 6(2). (1994)

[SWCH01] Sullivan, k., William, G., Cai, Y., Hallen, B.: The structure and value of modularity in software design. Journal SIGSOFT Software Engineering Notes. Vol. 26, number 5. ACM Press. Issn: 0163-5948. (2001) pp. 99–108

[T00] Temmerman, T.: Towards New Ways of Terminology Description, the sociocognitive approach. John Benjamins Publishing Company. Amsterdam. ISBN 9027223262. (2000)

[T96] de Troyer, O.: A Formalization of the Binary Object-Role Model based on Logic. Data & Knowledge Engineering 19, North-Holland Elsevier. (1996) pp. 1–37

[TB01] Tamma, V., Bench-Capon, T.: A conceptual model to facilitate knowledge sharing in multi-agent systems. Proceedings of the Autonomous Agents 2001 Workshop on Ontologies in Agent Systems (OAS'01). Montreal. May (2001) pp. 69–76

[TM95] de Troyer, O., Meersman, R.: A Logic Framework for a Semantics of Object-Oriented Data Modelling. In: Papazoglou, M.P. (eds.): Proceedings of 14th International Conference Object-Orientation and Entity-Relationship Modelling (OO-ER'95), Lecture Notes in Computer Science 1021, Springer. (1995) pp. 238–249

[TSC01] Tzitzikas, Y., Spyratos, N., Constantopoulos, P.: Mediators over Ontology-based Information Sources. Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01). (2001)

[TTN97] Takaai, M., Takeda, H., Nishida, T.: Distributed ontology development environment for multi-agent systems. Working Notes for AAAI'97. Spring Symposium Series on Ontological Engineering. (1997) pp. 149–153

[U01] Uitermark, H.: Ontology Based Geographic Data Set Integration. PhD Thesis, Twente University. (2001)

[U96] Uschold, M.: Building ontologies: Towards a Unified Methodology. Proceedings of Expert Systems, the 16th Annual Conference of the British Computer Specialist Group of Expert Systmes (AIAI-TR'97). Cambridge. December (1996)

[UG96] Uschold, M. and Gruninger, M: Ontologies: principles, methods and applications. Knowledge Engineering Review, vol. 11, no. 2 (1996)

-D

[V82] Van Griethuysen, J.J., (Eds.): Concepts and Terminology for the Conceptual Schema and Information Base. International Standardization Organization, Publication No. ISO/TC97/SC5- N695. (1982)

[V83] Vermeir D.: Semantic Hierarchies and Abstraction in Conceptual Schemata. Journal of Information Systems. Vol. 8, No. 2. (1983) pp. 117–124

[V98] Vossen, P. (eds.): EuroWordNet: A Multilingual Database with Lexical Semantic Networks. Kluwer Academic Publishers, Dordrecht. (1998)

[VB82] Verheijen, G., van Bekkum, P.: NIAM, aN Information Analysis Method. In: Olle, T.W., Sol, H., Verrijn-Stuart, A. (eds.), IFIP Conference on Comparative Review of Information Systems Methodologies, North-Holland. (1982) pp. 537–590

[VDM04] Verheyden, P., De Bo, J., Meersman, R.: Semantically unlocking database content through ontology-based mediation . In, Bussler C. & Tannen V.,(eds.), Proceedings of the 2nd Workshop on the Semantic Web and Databases (in conjuction with the 30th International Conference on Very Large Databases), LNCS 3372, Springer Verlag. (2004)

[VDZ04] Verlinden R., De Bo J., Zhao G.: Ontology Alignment and Merging Components. Deliverable 5.1.3. FF-Poirot project. IST – EU, 5th Framework (IST-2001-38248). (2004)

[VH91] Ventrone, V., Heiler, S.: Semantic Heterogeneity as a Result of Domain Evolution. SIGMOD Record 20(4). (1991) pp. 16–20

[VKMND04] Verbert, K., Klerkx, J., Meire, M., Najjar, J., Duval, E.: Towards a Global Component Architecture for Learning Objects: An Ontology Based Approach. Proceeding of On the Move to Meaningful

Internet Systems: OTM 2004 Workshops, LNCS 3292, Cyprus. (2004) pp. 713–722

[VOS03] Volz R., Oberle D., Studer R.: Views for light-weight web ontologies. Proceedings of the ACM Symposium on Applied Computing (SAC'03). (2003)

[VS03] Vassileva, B., Scoggins, P.: Consumer Complaint Forms: An Assessment, Evaluation and Recommendations for Complaint Categorization. Technical report, CCForm Project (IST-2001-34908), 5th framework. Brussels (2003)

[W02] Welty, C.: Ontology-Driven Conceptual Modeling. Invited talk at the Fourteenth International Conference on Advanced Information Systems Engineering (CAiSE), Toronto, Canada. (2002)

[W90] Wintraecken, J.J.V.R.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer. (1990)

[W97] Wiederhold, G.: Value-added Mediation in Large-Scale Information Systems. DS-6. (1995) pp. 34–56

[W98] Weinstein, P.C.: Ontology-Based Metadata: Transforming the MARC Legacy. ACM Digital Libraries. Pittsburgh, USA. (1998)

[WF99] Welty, C., Ferrucci, D.: A Formal Ontology for Re-Use of Software Architecture Documents. Proceedings of The 1999 International Conference on Automated Software Engineering. IEEE Computer Society Press. October (1999) pp. 259–262

[WG01] Welty, C., Guarino, N.: Support for Ontological Analysis of Taxonomic Relationships. Journal of Data and Knowledge Engineering. 39(1). October (2001) pp. 51–74

[WG03] Welty, C., Guarino, N.: An Overview of OntoClean. In: Staab, S., Studer, R., (eds.): The Handbook of Ontologies. Springer Verlag. (2003)

[WJ99] Welty, C., Jessica, J.: An Ontology for Subject. J. Data and Knowledge Engineering. 31(2). Elsevier. (1999) pp. 155–181

[WSG+04] Wache H., Serafino L., Garcia Castro R., Groot P., Jarrar M., Kompatsiaris Y., Maynard D., Pan J., Roelofsen F., Spaccapietra S., Stamou G., Tamilin A, Zaihrayeu I.: Scalability - State of the Art. Deliverable D2.1.1, EU-IST Network of Excellence (NoE) IST-2004-507482 (KWEB). Luxemburg (2005)

[WSW99] Wand, Y., Storey, V., Weber, R.: An Ontological Analysis of the relationship Construct in Conceptual Modelling. ACM Transactions on Database Systems, Vol. 24, No. 4. (1999) pp. 494–528

[ZD04] Ziegler, P., Dittrich, K.: User-Specific Semantic Integration of Heterogeneous Data: The SIRUP Approach. In: M. Bouzeghoub, C. Goble, V. Kashyap, S. Spaccapietra, (eds.): Proceeding of the International Conference on Semantics of a Networked World. LNCS, Springer, Paris, France. June (2004) pp. 14–44 .

[ZKK+04] Zhao G., Kingston J., Kerremans K., Coppens F., Verlinden R., Temmerman R. & Meersman R., Engineering an Ontology of Financial Securities Fraud. In, Meersman R., Tari Z. et al.,(eds.), On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, LNCS 3292, pp. 605 - 620, 2004. Springer Verlag.