



NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — “Semantic-based knowledge and content systems”

D1.1.5 Updated Version of the Networked Ontology Model

Deliverable Co-ordinator: Peter Haase

Deliverable Co-ordinating Institution: Universität Karlsruhe (TH) (UKARL)

Other Authors: Raul Palma (UPM), Mathieu d’Aquin (OU)

Document Identifier:	NEON/2009/D1.1.5/1.1	Date due:	28.02.2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	06.09.2009
Project start date	March 1, 2006	Version:	1.1
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

<p>Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk</p>	<p>Universität Karlsruhe – TH (UKARL) Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB D-76128 Karlsruhe Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de</p>
<p>Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es</p>	<p>Software AG (SAG) Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com</p>
<p>Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com</p>	<p>Institut 'Jožef Stefan' (JSI) Jamova 39 SL-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si</p>
<p>Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat</p>	<p>University of Sheffield (USFD) Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham</p>
<p>Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de</p>	<p>Consiglio Nazionale delle Ricerche (CNR) Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it</p>
<p>Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de</p>	<p>Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org</p>
<p>Atos Origin S.A. (ATOS) Calle de Albarraçín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parietelobo@atosorigin.com</p>	<p>Laboratorios KIN, S.A. (KIN) C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es</p>

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed writing parts of this document:

- UKARL
- UPM
- INRIA
- OU
- CNR

Change Log

Version	Date	Amended by	Changes
0.1	31.1.2009	Peter Haase	Initial creation of document based on D1.1.2
0.2	14.2.2009	Raul Palma	Updates to OMV
0.3	15.2.2009	Peter Haase	Updates for OWL 2
0.4	18.2.2009	Yiorgos Trimponias	E-Connections
0.5	19.2.2009	Peter Haase	Updates to module metamodel
0.9	28.2.2009	Peter Haase	Proof reading and final changes
1.0	20.3.2009	Peter Haase	Final changes after reviewer comments
1.1	05.08.2009	Andreas Harth	Clarified introduction and conclusion

Executive Summary

In this deliverable we present an updated version of the NeOn networked ontology model for representing and managing relations between multiple networked ontologies.

To achieve flexibility and applicability, we define the networked ontology model using a metamodeling approach. The metamodeling features of Model Driven Architecture provide the means for the specification of modeling languages in a standardized, platform independent manner. In short, the Meta Object Facility (MOF) provides the language for creating metamodels, UML defines the language for creating models corresponding to specific metamodels. Defining the networked ontology model in terms of a MOF compliant metamodel yields a number of advantages, including (1) interoperability with software engineering approaches, (2) model transformations, to support the automated generation of APIs, exchange syntaxes, etc., (3) reuse of UML for modeling, and (4) independence from particularities of specific formalisms, enabling the ability to support currently competing formalisms (e.g. in the case of mapping languages), for which no standard exists yet.

The metamodel consists of individual modules for the individual aspects of networked ontologies. The main modules are: (1) a metamodel for the OWL ontology language, (2) a rule metamodel, (3) a metamodel for ontology mappings, and (4) a metamodel for modular ontologies. The metamodel is grounded by translations to specific logical formalisms that define the semantics of the networked ontology model.

The deliverable extends the previous version of the networked ontology model presented in D1.1.2. We expect that the extended models presented here will be subsequently propagated to and implemented in the tool infrastructure. With respect to the metamodel, the changes over D1.1.2 describing a previous networked ontology model are mainly:

- We provide a metamodel for the OWL 2 language.
- We provide a metamodel for link properties that allow relating ontologies with disjoint domains within a network.
- We present a refined version of the metamodel for modular ontologies, aligned with the new OWL 2 metamodel.

Additionally, we provide an updated version of the Ontology Metadata Vocabulary OMV, our vocabulary to describe ontology metadata in a network of ontologies. Metadata here refers to data *about* the actual ontologies, as opposed to the content contained *in* an ontology, e.g. the provenance and authorship of an ontology, but also dependencies from other ontologies, etc. Metadata plays an important role in describing and managing the networked ontologies. Its purpose is to be able to clarify the relations between the available ontologies so that they are easy to search, to characterize and to maintain.

Contents

I Foundations	13
1 Introduction	14
1.1 The NeOn Big Picture	14
1.2 Networked Ontologies	15
1.2.1 The current situation	15
1.2.2 Objective and Definition of the NeOn networked ontology model	17
1.2.3 A metamodel of networked ontologies	17
1.2.4 Networked Ontology Metadata	18
1.2.5 Alignment with C-ODO	18
1.3 Overview of the Deliverable	19
2 Ontology Languages	20
2.1 Web Ontology Language (OWL)	21
2.2 Semantic Web Rule Language (SWRL)	22
2.3 Frame Logic (F-Logic)	22
2.4 OWL Ontology Mapping Languages	23
II The NeOn Metamodel for Networked Ontologies	24
3 Metamodeling for Ontologies	25
3.1 Metamodeling with MOF	25
3.1.1 The Four-Layer Architecture of MOF	25
3.1.2 Available Constructs in MOF	26
3.1.3 MOF vs. UML	27
3.2 A Networked Ontology Metamodel	27
3.2.1 Design Considerations	28
3.2.2 Advantages of the Networked Ontology Metamodel	28
3.2.3 Semantics of the Metamodel	29
4 The OWL Metamodel	30
4.1 Ontologies and Annotations	30
4.2 Entities and Data Ranges	31
4.3 Class Descriptions	33
4.4 OWL Axioms	38
4.5 Class Axioms	38
4.6 Object Property Axioms	39

4.7	Data Property Axioms	41
4.8	Assertions	42
4.9	Annotation Axioms	43
5	The Rule Metamodel	46
5.1	A Metamodel for SWRL Rules	46
5.1.1	Rules	46
5.1.2	Predicate Symbols	47
5.1.3	Terms	48
5.2	A Metamodel for F-Logic Rules	49
5.2.1	F-Logic Programs	50
5.2.2	Terms	50
5.2.3	Formulas	51
5.2.4	Rules and Queries	52
5.2.5	Logical Connectives	55
5.2.6	Logical Quantifiers	56
5.2.7	F-Atoms and F-Molecules	57
5.2.8	P-Atoms	59
6	The Mapping Metamodel	62
6.1	A Common MOF-based Metamodel Extension for OWL Ontology Mappings	62
6.1.1	Mappings	62
6.1.2	Queries	65
6.2	OCL Constraints for C-OWL	66
6.3	OCL Constraints for DL-Safe Mappings	67
7	The Metamodel for Modular Ontologies	69
7.1	Design Considerations	69
7.1.1	Existing Formalisms for Ontology Modularization	69
7.1.2	Requirements for a Module Definition Languages	72
7.1.3	Abstract Definition and Notation	73
7.1.4	Metamodel	74
8	The Extended OWL Metamodel for Linking Ontologies with \mathcal{E}-Connections	76
8.1	Motivation of \mathcal{E} -Connections	76
8.2	Preliminaries	77
8.3	The Link Property Entity	77
8.4	Class Expressions	77
8.5	Link Property Axioms	80
8.6	Assertions	82
III	Metadata for Networked Ontologies	84
9	The NeOn Ontology Metadata Vocabulary	85
9.1	Preliminary considerations	86
9.1.1	Metadata Categories	86

9.2	Ontology Metadata Requirements	87
9.3	Ontology Metadata Vocabulary	88
9.3.1	Core and Extensions	88
9.3.2	Ontological representation	88
9.3.3	Identification, Versioning and Location	88
9.3.4	OMV core metadata entities	90
10	Conclusion	93
10.1	Summary	93
A	Naming Conventions	94
A.1	Delimiters and capitalization	94
A.2	Property naming	94
A.3	Singular form	94
A.4	Additional considerations	94
B	OMV Core Ontology	96
B.1	Ontology	97
B.2	OntologyType	109
B.2.1	Pre-defined ontology types	109
B.3	LicenseModel	111
B.3.1	Pre-defined license models	113
B.4	OntologyEngineeringMethodology	114
B.5	OntologyEngineeringTool	116
B.5.1	Pre-defined ontology languages	118
B.6	OntologySyntax	119
B.6.1	Pre-defined ontology syntaxes	121
B.7	OntologyLanguage	122
B.7.1	Pre-defined ontology languages	124
B.8	KnowledgeRepresentationParadigm	125
B.8.1	Pre-defined knowledge representation paradigms	127
B.9	FormalityLevel	128
B.9.1	Pre-defined formality levels	128
B.10	OntologyTask	129
B.10.1	Pre-defined ontology tasks	129
B.11	OntologyDomain	132
B.12	Party	133
B.13	Person	136
B.14	Organisation	138
B.15	Location	139
	Bibliography	141

List of Tables

B.1 Class: Ontology	97
B.2 Property: URI	97
B.3 Property: name	97
B.4 Property: acronym	97
B.5 Property: description	98
B.6 documentation	98
B.7 reference	98
B.8 notes	98
B.9 Property: naturalLanguage	99
B.10 Property: keywords	99
B.11 Property: keyClasses	99
B.12 Property: status	99
B.13 Property: creationDate	100
B.14 Property: modificationDate	100
B.15 Property: hasContributor	101
B.16 Property: hasCreator	101
B.17 Property: usedOntologyEngineeringTool	101
B.18 Property: usedOntologyEngineeringMethodology	101
B.19 Property: conformsToKnowledgeRepresentationParadigm	102
B.20 Property: endorsedBy	102
B.21 Property: hasDomain	102
B.22 Property: isOfType	102
B.23 Property: designedForOntologyTask	103
B.24 Property: hasFormalityLevel	103
B.25 Property: knownUsage	103
B.26 Property: hasOntologyLanguage	103
B.27 Property: hasOntologySyntax	104
B.28 Property: resourceLocator	105
B.29 Property: version	105
B.30 Property: hasLicense	106
B.31 Property: useImports	106
B.32 Property: hasPriorVersion	106
B.33 Property: isBackwardCompatibleWith	107
B.34 Property: isIncompatibleWith	107
B.35 Property: numberOfClasses	107

B.36 Property: numberOfProperties	107
B.37 Property: numberOfIndividuals	108
B.38 Property: numberOfAxioms	108
B.39 Class: OntologyType	109
B.40 Property: name	109
B.41 Property: acronym	109
B.42 Property: description	109
B.43 documentation	110
B.44 definedBy	110
B.45 Class: LicenseModel	111
B.46 Property: name	111
B.47 Property: acronym	111
B.48 Property: description	111
B.49 documentation	112
B.50 specifiedBy	112
B.51 Class: OntologyEngineeringMethodology	114
B.52 Property: name	114
B.53 Property: acronym	114
B.54 Property: description	114
B.55 documentation	115
B.56 developedBy	115
B.57 Class: OntologyEngineeringTool	116
B.58 Property: name	116
B.59 Property: acronym	116
B.60 Property: description	116
B.61 documentation	117
B.62 developedBy	117
B.63 Class: OntologySyntax	119
B.64 Property: name	119
B.65 Property: acronym	119
B.66 Property: description	119
B.67 documentation	120
B.68 developedBy	120
B.69 Class: OntologyLanguage	122
B.70 Property: name	122
B.71 Property: acronym	122
B.72 Property: description	122
B.73 Property: documentation	123
B.74 Property: developedBy	123
B.75 Property: supportsRepresentationParadigm	123
B.76 Property: hasSyntax	123
B.77 Class: KnowledgeRepresentationParadigm	125
B.78 Property: name	125
B.79 Property: acronym	125

B.80 Property: description	125
B.81 documentation	126
B.82 Property: specifiedBy	126
B.83 Class: FormalityLevel	128
B.84 Property: name	128
B.85 Property: description	128
B.86 Class: OntologyTask	129
B.87 Property: name	129
B.88 Property: acronym	129
B.89 Property: description	129
B.90 documentation	130
B.91 Class: OntologyDomain	132
B.92 Property: URI	132
B.93 Property: name	132
B.94 Property: isSubDomainOf	132
B.95 Class: Party	133
B.96 Property: isLocatedAt	133
B.97 Property: develops	133
B.98 Property: specifies	133
B.99 Property: defines	134
B.100 Property: endorses	134
B.101 Property: hasAffiliatedParty	134
B.102 Property: createsOntology	134
B.103 Property: contributesToOntology	135
B.104 Class: Person	136
B.105astName	136
B.106irstname	136
B.107eMail	136
B.108Property: phoneNumber	137
B.109axNumber	137
B.110sContactPerson	137
B.111Class: Organisation	138
B.112Property: name	138
B.113Property: acronym	138
B.114hasContactPerson	138
B.115Class: Location	139
B.116Property: state	139
B.117Property: country	139
B.118Property: city	139
B.119Property: street	140

List of Figures

1.1	Relationships between different workpackages in NeOn	15
1.2	The current state of networked ontologies on the web. Dashed relations are in fact not explicitly recorded on the web but can be inferred from user actions.	16
1.3	The NeOn networked ontology model as a MOF-metamodel	18
1.4	The localization of the NeOn networked ontology metadata (M) with regard to the actual ontologies (O). Rectangular boxes refer to machines	19
3.1	OMG's four-layer metamodel hierarchy	26
3.2	Modules of the Networked Ontology Metamodel and possible groundings in ontology languages	27
4.1	OWL metamodel: ontologies	31
4.2	OWL metamodel: annotations	31
4.3	OWL metamodel: entities	32
4.4	OWL metamodel: declarations	32
4.5	OWL metamodel: object property expressions	33
4.6	OWL metamodel: data property expressions	33
4.7	OWL metamodel: data ranges	34
4.8	OWL metamodel: propositional connectives	35
4.9	OWL metamodel: object property restrictions	36
4.10	OWL metamodel: object property cardinality restrictions	37
4.11	OWL metamodel: data property restrictions	37
4.12	OWL metamodel: data property cardinality restrictions	38
4.13	OWL metamodel: axioms	38
4.14	OWL metamodel: class axioms	39
4.15	OWL metamodel: object property axioms - part 1	40
4.16	OWL metamodel: object property axioms - part 2	41
4.17	OWL metamodel: object property axioms - part 3	41
4.18	OWL metamodel: data property axioms - part 1	42
4.19	OWL metamodel: data property axioms - part 2	42
4.20	OWL metamodel: Assertions - part 1	43
4.21	OWL metamodel: Assertions - part 2	44
4.22	OWL metamodel: Assertions - part 3	45
4.23	Annotation Axioms	45
5.1	SWRL metamodel extension: rules	47
5.2	SWRL metamodel extension: predicate symbols	48

5.3	SWRL metamodel extension: terms	48
5.4	F-Logic metamodel: ontologies	50
5.5	F-Logic metamodel: terms	51
5.6	F-Logic metamodel: formulas	52
5.7	F-Logic metamodel: rules and queries	53
5.8	F-Logic metamodel: logical connectives	56
5.9	F-Logic metamodel: logical quantifiers	57
5.10	F-Logic metamodel: F-molecules	59
5.11	F-Logic metamodel: F-molecule host objects	59
5.12	F-Logic metamodel: methods	60
5.13	F-Logic metamodel: P-atoms	61
6.1	OWL mapping metamodel: mappings	63
6.2	OWL mapping metamodel: queries	65
7.1	Metamodel extensions for ontology modules.	74
8.1	Extended metamodel: link property expressions	78
8.2	Extended metamodel: link property restrictions	78
8.3	Extended metamodel: link property cardinality restrictions	79
8.4	Extended metamodel: link property axioms - part 1	80
8.5	Extended metamodel: link property axioms - part 3	81
8.6	Extended metamodel: link property axioms - part 2	82
8.7	Extended metamodel: assertions	83
9.1	OMV overview	91

Part I

Foundations

Chapter 1

Introduction

1.1 The NeOn Big Picture

Next generation semantic applications will be characterized by a large number of ontologies, some of them constantly evolving. As the complexity of semantic applications increases, more and more knowledge will be embedded in applications, typically drawn from a wide variety of sources. This new generation of applications will thus likely rely on ontologies embedded in a network of already existing ontologies. Ontologies and metadata will have to be kept up to date when application environments and users' needs change. We argue that in this scenario it will become prohibitively expensive for people to directly adopt the current approach to semantic integration, where the expectation is to produce a single, globally consistent semantic model that serves the needs of application developers and fully integrates a number of pre-existing ontologies. In contrast to the current model, future applications will very likely rely on networks of contextualized ontologies, which are usually locally, but not globally consistent.

This report is part of the work performed in WP 1 on "Dynamics of Networked Ontologies". The goal of this work package is to develop an integrated approach for the evolution process of networked ontologies and related metadata. For the individual phases of the process we will develop new methods that consider the complex relationships in a network of ontologies. These include dependencies, mappings, different versions and also take possible inconsistencies into account. Where possible, these methods will be supported by automatic or semi-automatic approaches based on natural language processing (NLP), data mining and machine learning techniques, and the corresponding tools will be developed to support the evolution process. Specific goals in this workpackage include support for:

1. representing, managing and interpreting dependencies between multiple networked ontologies
2. evolution of networked ontologies in exploiting various models of change propagation, which have different applicabilities depending on the model of coordination and control
3. maintaining partial/local consistency of a set of networked ontologies, which might not be globally consistent
4. evolving metadata along with changing ontologies and predicting future structural changes in ontologies.

In order to achieve the goals mentioned above it is crucial that a toolkit designed within NeOn offers a basic set of functionalities, which are also essential for the use cases. These are: (a) Integration of a large number of heterogeneous information sources (b) Maintenance of ontologies to reflect changes in the domain (c) Consensus forming in a decentralized scenario.

As shown in Figure 1.1, WP1 belongs to the central part of the research and development WPs in NeOn. The tasks of WP1 are heavily inter-related with other work packages. The most fundamental contribution of WP1, and in particular Task 1.1, is to provide the networked ontology model and its related dynamic aspects

to other work package partners. As a consequence, a major objective in the definition of the networked ontology model is the consideration of requirements from other work packages. We will closely collaborate with the case study partners to apply our technologies within the case studies.

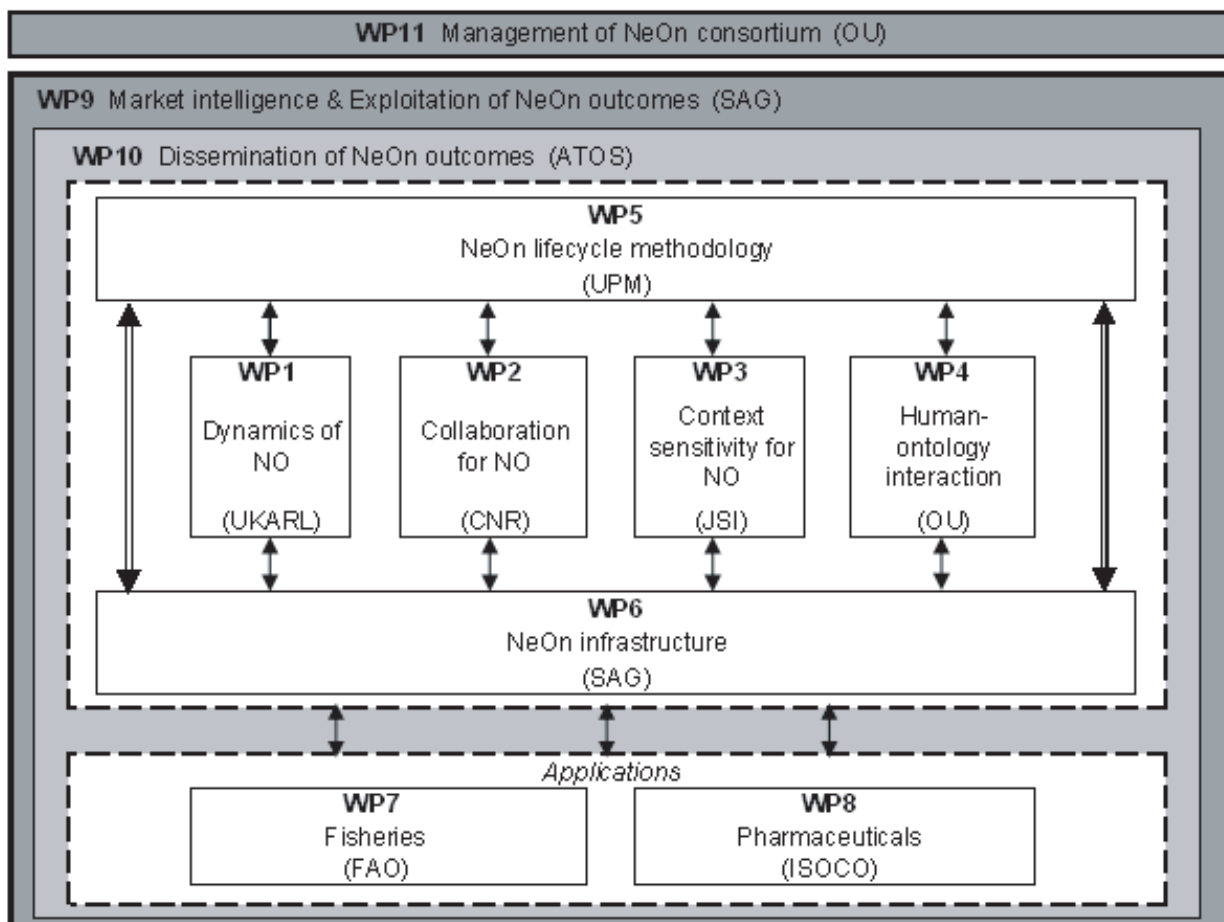


Figure 1.1: Relationships between different workpackages in NeOn

1.2 Networked Ontologies

The goal of this deliverable is to define and formalize the notion of networked ontologies in order to make it applicable throughout the NeOn project. Before formally defining the model, we will first informally discuss the notion of a networked ontology.

1.2.1 The current situation

The OWL ontology language is now well established as the standard ontology language for representing knowledge on the web. At the same time, rule languages, such as F-Logic, have shown their practical applicability in industrial environments. Often, ontology-based applications require features from both paradigms—the description logics and the rule paradigm—but their combination remains difficult. This is not only due to the semantic impedance mismatch [HPPSH05], but already because of the disjoint landscape in ontology engineering tools that typically support either the one or the other paradigm.

Additionally, to address distributed and networked scenarios as envisioned in NeOn, current ontology languages lack a number of features to explicitly express the relationships between ontologies and their el-

ements. These features include in particular formalisms for expressing *modular ontologies* and *mappings* (also called alignments) between ontologies that are heterogeneous on various respects.

Finally, in the current situation we lack the means to express information *about* ontologies and the relationships between them. Ontologies are distributed over the web, sometimes available directly, sometimes hidden within corporate networks (see Figure 1.2¹). These ontologies are related to each others, but this remains difficult to assess:

- some are simple copies of other ones (it is hard to know which one is the master copy);
- some are versions of others (it is hard to know which one came first);
- some are used jointly with others (this information is hidden in applications);
- some are imported by others (this can be found through owl:imports).

Moreover, the ontologies have different characteristics:

- they are written in different languages and with different language variants;
- they use labels in different natural languages;
- they are built for different purposes.

In consequence, each application developer has to manually assemble ontologies, modify, import and export them, resulting in yet other disconnected, un-contextualized ontologies. Each time one of these ontologies is changed, at best, the developer will propagate the changes, at worst, they will not be taken into account at all.

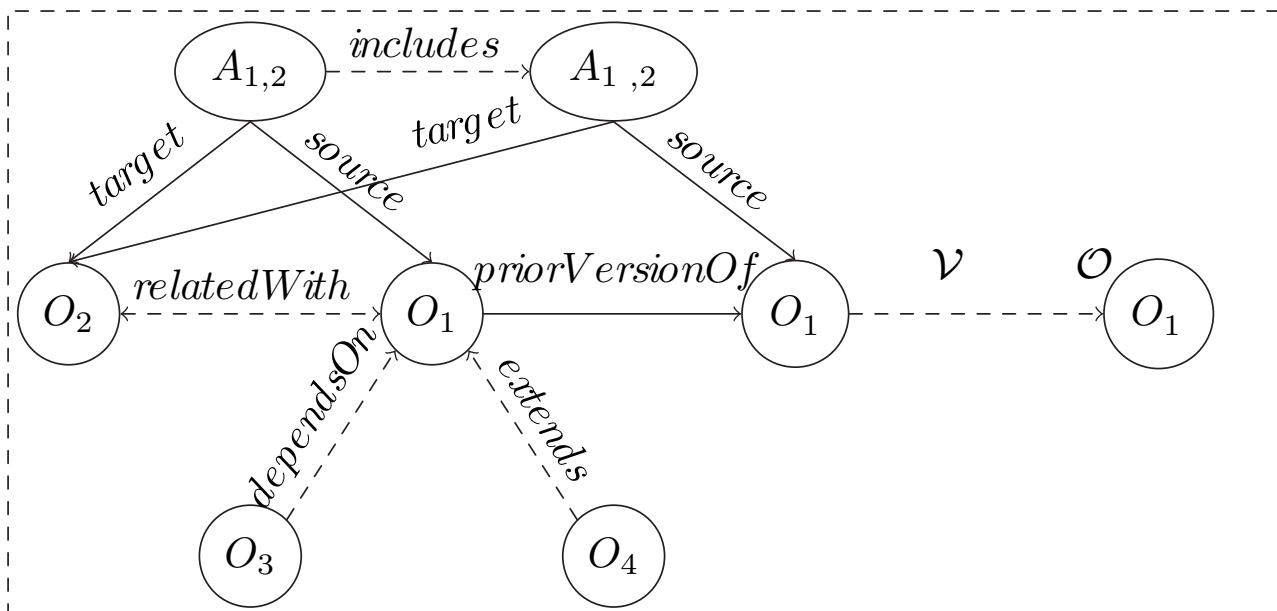


Figure 1.2: The current state of networked ontologies on the web. Dashed relations are in fact not explicitly recorded on the web but can be inferred from user actions.

This state of affairs constitutes a threat for ontology-based application development by incurring costs of sorting out the available material as well as lack of support for maintaining ontologies over their full life-cycle.

¹In this figure, O denotes ontologies, A denotes mappings (alignments).

1.2.2 Objective and Definition of the NeOn networked ontology model

The purpose of the NeOn networked ontology model is not to be a completely new language for expressing ontologies on the web. Its purpose is to define an integrated model of an ontology language covering the relevant existing ontology languages and extending them with primitives to express the relationships in a network of ontologies. For that purpose, it will introduce – on top of existing languages (e.g. OWL Web Ontology Language [Mv03], F-Logic) – a model of the interrelations between ontologies. Because the goal of NeOn is to non-exclusively account for existing formalisms, it will attempt to reuse these legacy formalism as much as possible and refrain from defining new languages when this is not necessary.

Subsequently, if we talk about networked ontologies and networks of ontologies, we refer to the following definition:

Definition 1 *A Network of Ontologies is a collection of ontologies related together via a variety of different relationships such as mapping, modularization, version, and dependency relationships. We call the elements of this collection Networked Ontologies.*

In particular, we support ontologies that are expressed in either OWL 2² or F-Logic. In this deliverable, we define particular networking relationships in detail. These include: mappings between heterogeneous ontologies, linking ontologies with disjoint domains, and modular ontologies. In the following, we provide intuitive definitions of these relationships.

Definition 2 *A mapping between two ontologies is a set of mapping assertions representing semantic relations between ontology elements of these ontologies.*

While mappings require the existence of correspondences between two ontologies (and thus overlaps in their modeled domain), it may also be needed to relate ontologies that do not overlap (disjoint domains). For this we allow for the notion of linking ontologies using ϵ -connections:

Definition 3 *An ϵ -connection between two disjoint ontologies is a special link between them, which relates the domains of the two ontologies. They can be conveniently conceived as superroles relating the two domains and they can be used as constructors for complex concepts.*

Finally, ontologies can be related as modules:

Definition 4 *An ontology module is an ontology that exposes elements to be reused by other ontology modules (defined via an export interface and in turn imports elements from other ontology modules (defined via import interfaces). Additionally, an ontology module may import mappings to define the correspondences with elements in the imported ontology modules.*

A set of modular ontologies with their interrelationships and mappings make up a modular ontology space.

1.2.3 A metamodel of networked ontologies

In this deliverable, we describe the networked ontology model in terms of a metamodel. Metamodels are used for the specification of modeling languages in a standardized, platform independent manner. Furthermore, a grounding essentially specifies the bi-directional translation of the terms of the metamodel to those of the specific formalism.

The general idea of using a metamodel-based approach and UML profiles for this purpose is depicted in Figure 1.3: The metamodel for networked ontologies as well in MOF, i.e. it is defined in terms of the MOF meta-metamodel (explained in Detail in Chapter 3). The term metamodel is chosen, as a metamodel refers to model of a language, whereas the instances of the metamodel are referred to as models. In our case, models thus refer to the actual ontologies.

²The new version OWL 2 (on which this version of the metamodel is built) is an extension of OWL DL (on which the previous version of the metamodel was built). As such, the new version of the metamodel is backward compatible with the previous one.

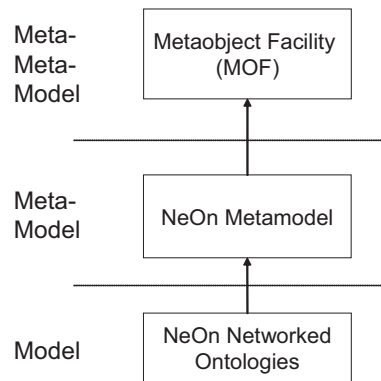


Figure 1.3: The NeOn networked ontology model as a MOF-metamodel

Semantics of the metamodel It should be noted that a metamodel is targeted mainly at a *structural specification* of a language, i.e. it is not intended to fully define the (logical) semantics of the language. The semantics is given by the bi-directional translation the metamodel to the underlying logical language. For example, the semantics of the OWL metamodel is given by the model theory of the underlying Description Logic. As such, the metamodel also does not aim at bridging the *semantic differences* between different ontology languages (OWL and F-Logic in particular). Yet, while the semantics of the languages is not directly known to a metamodelling-based system, the common representation within the same metamodelling framework allows translations between different languages based on so-called metamodel transformations. Here, translations are made between logical patterns of the languages, as for example shown in [Bro07] for the translation from F-Logic to OWL.

1.2.4 Networked Ontology Metadata

Additionally, we provide a vocabulary to describe ontology metadata in a network of ontologies. Metadata here refers to data *about* the actual ontologies, as opposed to the content contained *in* an ontology, e.g. the provenance and authorship of an ontology, but also dependencies from other ontologies, etc. Metadata plays an important role in describing and managing the networked ontologies. Its purpose is to be able to clarify the relations between the available ontologies so that they are easy to search, to characterize and to maintain. It aims at making explicit the virtual and implicit network of ontologies.

A NeOn enabled tool will be able to take advantage of the NeOn networked ontology metadata for selecting the resources appropriate for users' needs. It will also enhance maintainability by updating the resources used by the applications when these resources evolve and tracking the conflicts that can occur in such a case. Moreover it should be able to export metadata descriptions according to the specification of the networked ontology model so that other tools can take advantage of them.

As presented in Figure 1.4, the metadata about networked ontologies can reside anywhere with regard to the ontologies: it can be embedded within actual ontology files, expressed in separate files along with the ontologies, expressed on completely different servers (like independent annotations) or reside on the local machine with the ontologies.

1.2.5 Alignment with C-ODO

There is also a strong relationship with the C-ODO metamodel developed for collaborative ontology design as developed in WP2: C-ODO is already aligned with OMV and the metamodel of networked ontologies, i.e. important aspects of ontology design are described in terms of the networked ontology model. The current version of C-ODO Light is available at: <http://www.ontologydesignpatterns.org/cpont/codo/codolight.owl> The alignments with the OWL metamodel and OMV are available

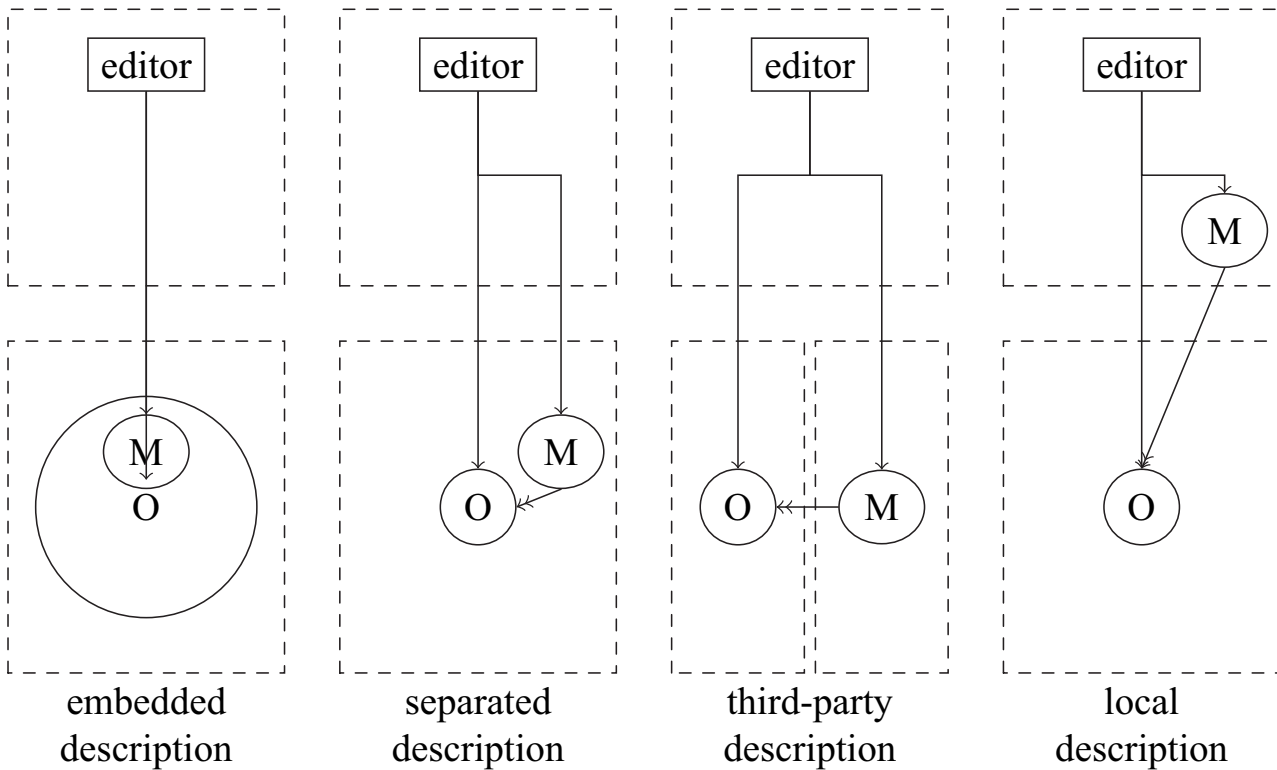


Figure 1.4: The localization of the NeOn networked ontology metadata (M) with regard to the actual ontologies (O). Rectangular boxes refer to machines

at <http://www.ontologydesignpatterns.org/cpont/codo/odm2codo.owl> and <http://www.ontologydesignpatterns.org/cpont/codo/omv2codo.owl>.

1.3 Overview of the Deliverable

In Chapter 3, we introduce the concept of metamodeling and its role for the definition of the networked ontology model. In the following chapters we introduce the specific modules of the metamodel: We start with the metamodel of OWL 2 in Chapter 4. We then present the metamodel for F-Logic and SWRL in Chapter 5, for mappings in Chapter 6, and for modularization in Chapter 7.

We then present OMV as a metadata ontology to capture information about networked ontologies in Chapter 9. Finally, in Chapter 10 we provide a summary and an outlook to subsequent deliverables.

Chapter 2

Ontology Languages

An ontology can be seen as a conceptual schema of a domain, representing the domain's data structure containing all the relevant entities and their relationships within that domain. Possibly, rules are defined on top of the ontology. Additionally, with the increasing use of ontologies, a new issue came up of aligning ontologies that describe the same domain.

OWL is the ontology language standardized by the World Wide Web Consortium and is very well adopted. F-Logic is another ontology language, which is well-known by quite some companies but, however, it was not intentionally built for ontologies but for deductive and object oriented databases. Next to the standard language OWL on which we focus our work, we incorporate F-Logic into our work.

Although OWL is very expressive, it is restricted to obtain decidability. This comes at the price that it can not express arbitrary axioms: the only axioms it can express are of a certain tree-structure. In contrast, decidable rule-based formalism such as function-free Horn rules are not so restricted in the use of axioms but lack some of the expressive power of OWL: they are restricted to universal quantification and lack negation in their basic form. To overcome the limitations of both approaches, several rule extensions for OWL have been heavily discussed [W3C05a] and the W3C initiated a working group for the definition of a Rule Interchange Format [W3C05b]. SWRL is one of the most prominent proposals for an extension of OWL with rules. DL-safe rules [MSS04] are a decidable subset of SWRL. As every DL-safe rule is also a SWRL rule, DL-safe rules are covered in our work. It should be noted that, although we consider SWRL as the most relevant rule extension for OWL in the context of our work, it is not the only rule language which has been proposed for ontologies. Other prominent alternatives for rule languages are mentioned in the W3C RIF charter, namely the Web Rule Language WRL [ABdB⁺05] and the rules fragment of the Semantic Web Service Language SWSL [GKM05]. These languages differ in their semantics and consequently also in the way in which they model implicit knowledge for expressive reasoning support. From this perspective, it could be desirable to provide particular support tailored to these specific rule languages. From the perspective of conceptual modeling, however, different rule languages appear to be very similar to each other. This opens up the possibility to reuse our outcome for SWRL by augmenting it with some features and language primitives which are not present in SWRL.

Often when ontologies are written in the same language, still they are modeled differently and can not be aligned automatically. Mappings between the heterogenous ontologies have to be defined by the user. For such mappings, we concentrate on OWL ontology mappings since OWL is the most prominent ontology language currently available. Our work covers all existing OWL ontology mapping formalisms we are currently aware of.

This chapter introduces the ontology languages that we support in our work. Firstly, Section 2.1 describes OWL, after which Section 2.2 introduces SWRL. Subsequently, the language F-Logic is discussed in Section 2.3. Finally, OWL ontology mapping languages are addressed in Section 2.4.

2.1 Web Ontology Language (OWL)

The Web Ontology Language OWL [Mv03, MPSP08], based on the earlier language DAML+OIL [vHPSH01], was standardized by the W3C in February, 2004. Since, it has become the most-accepted ontology language in the semantic web community and is supported by a constantly increasing number and range of applications. At the moment of writing this deliverable, a new version of OWL, OWL 2 [MPSP08], based on experiences during the language's first years, is undergoing the standardization process. OWL allows to build, publish and share ontologies. By describing the concepts of a domain as well as their relationships formally, the meaning of documents becomes machine-understandable. The formal semantics of OWL is derived from Description Logics (DL, [BCM⁺03]), an extensively researched knowledge representation formalism. Hence, most primitives offered by OWL can also be found in a Description Logic.

OWL is built on top of RDF [KC04] and extends the expressiveness of RDF and RDF Schema [BG04] by supporting additional vocabulary with a formal semantics.

The first version of OWL defined three sublanguages: OWL Lite, OWL DL and OWL Full. For these different language levels,

OWL Lite < OWL DL < OWL Full

holds in terms of expressivity.

OWL Full OWL Full allows expressions of higher order predicate logic, which results in the fact that OWL Full ontologies can be undecidable and hence OWL Full is impractical for applications that require complete reasoning procedures. Contrary to OWL Lite and OWL DL which impose restrictions on the use of the vocabulary, OWL Full supports the full syntactic freedom of RDF and can be seen as an extension of RDF. It has mainly been defined for compatibility with existing standards such as RDF.

OWL DL is equivalent to a decidable subset of first-order predicate logic. It closely corresponds to the $SHOIN(D)$ description logic and all language features can be reduced¹ to the primitives of the $SHOIN(D)$ logic. To allow the representation in this logic, OWL DL imposes several limitations on the set of supported language constructs. The limitations defined on OWL DL ease the development of tools and allow complete inference. For OWL DL, practical reasoning algorithms are known, and increasingly more tools support this or slightly less expressive languages.

OWL Lite is the smallest standardized subset of OWL Full, closely corresponding to the description logic known as $SHIF(D)$. It is mainly to support the design of classification hierarchies and simple constraints. Several language elements that are supported by OWL DL, are not available in OWL Lite, specifically number restrictions are limited to arities 0 and 1. Furthermore, the oneOf class constructor is missing. Other constructors such as class complement, which are syntactically disallowed in OWL Lite, can nevertheless be represented via the combination of syntactically allowed constructors [Vol04].

It holds that every legal OWL Lite ontology is a valid OWL DL ontology, and every valid OWL DL ontology is a valid OWL Full ontology. For our work, OWL DL would be most useful as it is the OWL dialect which is mostly used in practice as it is decidable although still very expressive. The new version OWL 2 on which our work is built, is an extension of OWL DL. OWL 2 provides additional Description Logic expressive power, moving from the $SHOIN(D)$ Description Logic that underlies OWL DL to the $SROIQ$ Description Logic [HKS06].^{2 3}

¹Some language primitives are shortcuts for combinations of primitives in the logic.

²The work in this deliverable is based on the working draft OWL 2 in *last call* state, available at the end of December, 2008.

³Throughout the rest of this deliverable, we are talking about OWL 2 whenever we refer to OWL.

OWL provides an elaborated set of constructs to define classes (concepts), properties and individuals (instances of one or more classes). We introduce the OWL 2 constructs while presenting our metamodel for OWL in Chapter 4.

The OWL 2 specifications provide the so-called functional-style syntax as a human-readable syntax. A syntax based on XML allows easy implementations of OWL 2. This XML exchange syntax is defined in the XML schema language [TBMM04]. For a full account of the available syntaxes, we refer the reader to [MPS08a]. Additionally, [MPS08b] provides a mapping from the OWL 2 functional-style syntax to the RDF exchange syntax [Bec04].

A formal semantics of OWL 2 is provided based on the principles for defining the semantics of description logics. The semantics is defined as a model-theoretic semantics [TV56] by interpreting the constructs of the functional-style syntax. The model-theoretic approach is an accepted paradigm for providing a formal account of meaning and entailment. To interpret OWL 2 ontologies serialized in the RDF syntax, they are translated into the functional-style syntax first. For a full account on the model-theoretic semantics of OWL, we refer to [MPSG08].

2.2 Semantic Web Rule Language (SWRL)

One of the most prominent proposals for an extension of OWL with rules is the Semantic Web Rule Language (SWRL, [HPSB⁺04]). SWRL proposes to allow the use of Horn-like rules together with OWL axioms. For instance, asserting that the combination of the *hasParent* and *hasBrother* properties implies the *hasUncle* property, is not possible in OWL but in SWRL.

The SWRL specifications, submitted to W3C in May 2004, include a high-level abstract syntax for Horn-like rules extending the OWL abstract syntax. An extension of the model-theoretic semantics from OWL provides the formal meaning for rules written in this abstract syntax. Moreover, next to the abstract syntax, SWRL allows an XML syntax based on the OWL XML presentation syntax, as well as an RDF concrete syntax based on the OWL RDF/XML exchange syntax. Both the different syntaxes and the model-theoretic semantics of SWRL are described in detail in [HPSB⁺04].

SWRL has a high expressive power but at the price of decidability. Moreover, it becomes undecidable as rules can be used to simulate role value maps ([SS89]). To balance the expressive power against the execution speed and termination of the computation, suitable subsets of the language can allow efficient implementations. The original SWRL specifications are built on OWL 1.0. The minor changes to adapt to OWL 2 are incorporated in our work.

We present the SWRL language constructs while explaining the metamodel in Chapter 5.1.

2.3 Frame Logic (F-Logic)

Frame Logic (F-Logic), for which we refer to [KLW95] for a full account, is a deductive, object-oriented and frame-based formalism. Originally, the language was developed for deductive and object-oriented databases in 1995. Later on, however, it has been applied for the implementation of ontologies. F-Logic provides constructs for defining declarative rules which infer new information from the available information. Furthermore, queries can be asked that directly use parts of the ontology. From a syntactic point of view, F-Logic is a superset of first-order logic. Currently a new version of FLogic is defined as a community process taking current developments from the semantic web area into account. Since these new developments are work in progress we focus on the current implementations of FLogic as addressed in [Ont06, FHK⁺97].

In contrast to OWL and SWRL, F-Logic does not have several syntactical representations defined. Again, we introduce the grammar and the object-model of F-Logic while presenting its metamodel in Chapter 5.2.

The F-Logic semantics is based on the fixpoint semantics of Datalog programs [UII88]. The evaluation starts with an empty object base, and rules and facts are evaluated iteratively (note that queries are not part of an

F-Logic program). When the rule body is valid in the actual object base with certain variable bindings, these bindings are propagated into the rule head. In this way, new information from rule heads is deduced due to closure properties, and inserted into the object base. Until no new information can be obtained anymore, rules are continuously evaluated. However, this is only the abstract view on the semantics. An F-Logic program is not necessarily executed in a bottom up fashion.

As with Datalog, the evaluation of a negation-free F-Logic program reaches a fixpoint which coincides with the unique minimal model of that program, which is defined as the smallest set of P- and F-atoms such that all closure properties and all facts and rules of the program are satisfied. As soon as a fixpoint is reached, the semantic of an F-Logic program is computed.

Additionally, F-Logic has a model-theoretic semantics. We refer to [KLW95] for a complete presentation of the F-Logic semantics.

2.4 OWL Ontology Mapping Languages

Since we believe OWL is the most important currently available ontology language, and since we chose OWL as the core of our work, we apply the logical restriction to focus on mappings between ontologies represented in OWL. In contrast to the area of ontology languages, where a de facto standard exists for representing and using ontologies, there is no agreement yet on the nature and the right formalism for defining mappings between ontologies.

OWL mapping formalisms are often based on non-standard extensions of the logics used to encode the ontologies. We focus on approaches that connect description logic based ontologies where mappings are specified in terms of logical axioms. This allows us to be more precise with respect to the nature and properties of mappings. At the same time, we cover all relevant mapping approaches [SSW05a] that have been developed that satisfy these requirements: C-OWL [BGvH⁺03], OIS (Ontology Integration System, [CDL01]), DL for II (DL for Information Integration, [CDL02]), and DL-Safe Mappings [HM05].

[SU05] researched the nature of ontology mappings and identified some general aspects of the above mapping approaches. We want to take these general aspects as a basis for our work, since we want to make sure that all these approaches are covered. Additionally, we provide specific support for C-OWL and DL-Safe Mappings by defining sets of metamodel constraints as well as language mappings for these concrete formalisms. For specific details on C-OWL and DL-Safe Mappings, we refer to Chapter 6, where we present their aspects along with the introduction of the metamodel.

Part II

The NeOn Metamodel for Networked Ontologies

Chapter 3

Metamodeling for Ontologies

3.1 Metamodeling with MOF

The Model Driven Architecture [MKW04, Fra03] is an initiative of the standardization organization OMG (Object Management Group) to support model-driven engineering of software systems based on the idea that modeling is a better foundation for developing and maintaining systems. MDA utilizes modeling as technique to raise the abstraction level and to effectively manage the complexity of systems. Models increase productivity, under the assumption that it is cheaper to manipulate the model, which is an abstraction of the system, than the system itself.

Using the MDA methodology, system functionality may first be defined as a technology neutral model through an appropriate modeling language. Then, using automated tools, this can be translated to one or more technology specific models. One of the fundamental components of MDA that is used in our work, is the Meta Object Facility.

When describing a universe of discourse in the real world, objects are being *classified* according to their common features. Furthermore, when referring to the objects, some of their properties which are not relevant may be left out. This step of describing objects with only those original properties that are of interest, and leaving out the irrelevant properties, is called *abstraction*. In addition, next to classification and abstraction, the classes of objects are grouped based on common features, a process called *generalization*.

These three steps of classifying, abstracting and generalizing take place in many contexts, but they are crucial in the modeling area. When formalizing the description of some domain in a model, these three steps are typically combined, and therefore performed at the same time. That is to say, a modeler looks at the real objects to be modeled, and abstracts away any information which seems not to be required. Simultaneously, the objects are classified into types and these types are put into generalization-hierarchies.

To support the MDA initiative, it is essential that designed models are commonly understood by all involved parties. This requires the ability to specify the meaning of a model. This is where *metamodels* come into play. A metamodel specifies the constructs of a modeling language, using the constructs that can be used in the description of a language. The Meta Object Facility [Obj06, Fra03] provides such a set of metamodeling constructs to define MOF-based metamodels as models of modeling languages.

We first introduce MOF using its four-layer architecture in Section 3.1.1. Next, Section 3.1.2 starting on page 26 introduces the available MOF constructs.

3.1.1 The Four-Layer Architecture of MOF

Before explaining some more details of MOF, let us look at the relationship between a metamodel and an instance model that represents the objects in the real world. A standardized terminology of OMG eases the communication about the different layers we just introduced, namely the *information layer*, the *model layer* and the *metamodel layer*. These are shown in the three bottom layers of Figure 3.1.

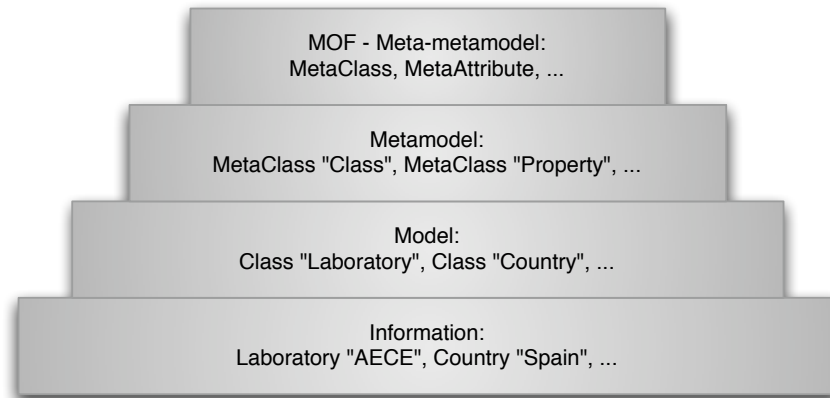


Figure 3.1: OMG's four-layer metamodel hierarchy

The undermost layer of this hierarchy encompasses the raw information to be described. For example, the figure contains information about a pharmaceutical laboratory called AECE and about the country Spain, in which the laboratory is located. One layer above, the model layer contains the definition of the required structures, like the classes used for classifying information. Thus in the example, the classes Laboratory and Country are defined. If these structures are combined, they describe the model for the given domain. The metamodel on the third layer defines the terms in which the model is expressed. In our example, we would state that models are expressed with classes and properties by instantiating the respective metaclasses. Important to mention is that the architecture allows models to span more than one of the lower levels. Ontologies do not have a clear separation between model and information. Therefore, in our work, the two bottom layers of Figure 3.1 are combined into one layer.

Finally, the figure also presents the fourth layer, the meta-metamodel layer called the MOF layer. It is the meta-metadata defining the modeling constructs that can be used to define and manipulate a set of interoperable metamodels on the level below. The MOF layer contains only the simplest set of concepts for models and metamodels, and captures the structure and semantics of arbitrary metamodels. Note that the top MOF layer is "hard wired" in the sense that it is fixed, while the other layers are flexible and allow to express various metamodels such as the existing UML metamodel or our metamodel for ontologies, rules and ontology mappings.

The model-driven framework supports any kind of metadata and allows new kinds to be added as required. Clearly, the MOF layer plays a crucial role in this four-layer metamodel hierarchy of OMG. It allows to define, manipulate and integrate metamodels and models in a platform-independent manner.

3.1.2 Available Constructs in MOF

The set of constructs provided by MOF is a simple set of concepts, though powerful enough for capturing the static structure of a model. The five MOF-concepts are represented as metaconcepts in Figure 3.1 and can define the abstract syntax of a language:

- types (classes, primitive types, and enumerations),
- generalization,
- attributes,
- associations, and
- operations.

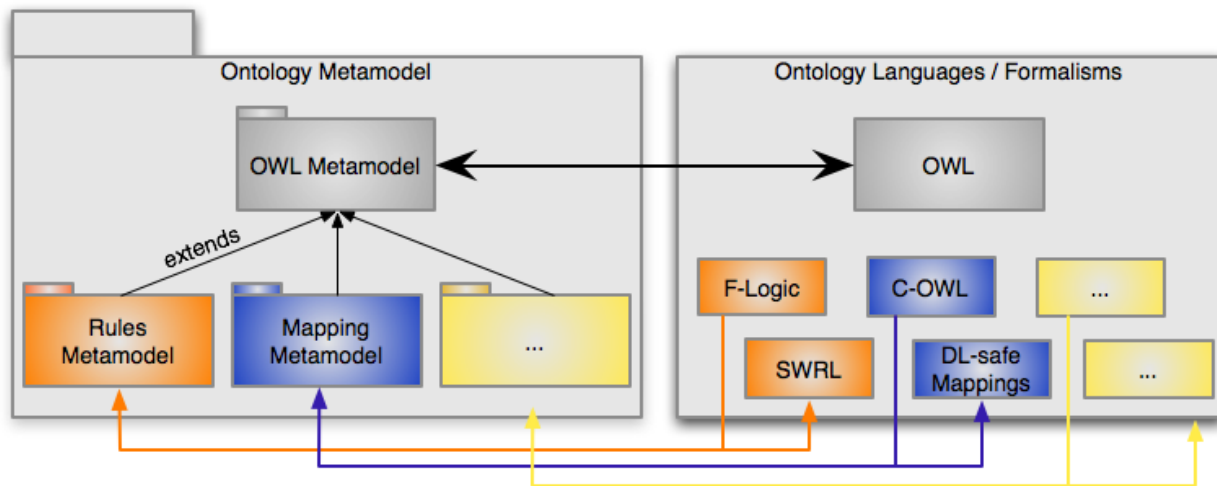


Figure 3.2: Modules of the Networked Ontology Metamodel and possible groundings in ontology languages

Ecore [BGS⁺03] has emerged as a quasi standard for the definition of metamodels and we provide our metamodels in the Ecore-format. Also, throughout the years, OMG has striven for a unification of MOF and UML ([Gro05], [Obj06]). This makes UML also suitable as a graphical notation for MOF metamodels. Moreover, when UML is used to build MOF metamodels, such specifications are not merely UML diagrams. Instead, MOF borrows object-oriented class-modeling constructs from UML and presents them for describing the abstract syntax of modeling constructs. Thus, MOF metamodels can look like UML class models, and one can use UML class-modeling tools to display them. We present our metamodels also in UML so that they are easily understandable for the reader.

3.1.3 MOF vs. UML

It should be noted that a MOF specification is not merely a UML diagram. Instead, MOF borrows OO class-modeling constructs from UML and presents them as the common means for describing the abstract syntax of modeling constructs. Thus, MOF metamodels look like UML class models, and one can use UML class-modeling tools to create them.

3.2 A Networked Ontology Metamodel

The ontology metamodel as well as a UML profile are grounded in MOF in that they are defined in terms of the MOF meta-metamodel. The UML profile mechanism is an extension mechanism to tailor UML to specific application areas. UML profiles define a visual notation for optimally supporting the specification of networked ontology models. This visual syntax is based on the metamodel. Mappings in both directions between the metamodel and the profile have to be established.

However, the OWL ontology metamodel is just one part of the networked ontology metamodel. The metamodel consists of several modules. The core module, i.e. the OWL metamodel, is extended by different modules that provide additional features, e.g. rules or mappings. In many application scenarios, only particular aspects of the networked ontology model are needed. In these cases, only the relevant modules need to be supported and used.

While the OWL ontology metamodel has a direct grounding in the OWL ontology language, the extensions have a generic character in that they are formalism independent and allow a grounding in different formalisms.

3.2.1 Design Considerations

Modularization The metamodel consists of several modules. The core module, i.e. the OWL metamodel, is extended by different modules that provide additional features, e.g. modularization, mappings, etc. In many application scenarios, only particular aspects of the networked ontology model are needed. In these cases, only the relevant modules need to be supported and used.

Compatibility with standards In terms of the metamodel, two aspects of standards are relevant: The first aspect relates to the fact that we are using a standard formalism—namely the Meta Object Facility—to describe the metamodel. The second aspect relates to the metamodel of networked ontologies itself: A major design goal is compatibility with existing ontology languages. With the Web Ontology Language OWL we have a standard for representing ontologies, therefore we provide a metamodel of OWL directly, with a one-to-one translation. For the other aspects of networked ontologies (mappings, rules, ...) no such standards exist yet. In favor of general applicability we therefore provide generic metamodels for these extensions that allow translations to different formalisms, as shown in Figure 3.2.

3.2.2 Advantages of the Networked Ontology Metamodel

Defining the ontology model in terms of a MOF compliant metamodel yields a number of advantages:

1. Interoperability with software engineering approaches In order for semantic technologies to be widely adopted by users and to succeed in real-life applications, they must be well integrated with mainstream software trends. This includes in particular interoperability with existing software tools and applications to put them closer to ordinary developers. MDA is a solid basis for establishing such interoperability. With the ontology model defined in MOF, we can utilize MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels.

2. Standardized model transformations MOF specifications are independent of particular target platforms (e.g. programming languages, concrete exchange syntaxes, etc.). Industry standardized mappings of the MOF to specific target languages or formats can be used by MOF-based generators to automatically transform the metamodel's abstract syntax into concrete representations based on XML Schema [TBMM04], Java, etc. For example, using the MOF-Java mapping, it is possible to automatically generate a Java API for a MOF metamodel.

3. Custom model transformations As MOF-based metamodels are all built using the same standardized constructs, mappings between them can be defined in a straightforward way. Based on these mappings between the metamodels, automatic transformations between models of the concerning languages can be performed.

4. Reuse of UML for modeling With respect to interoperability with other metamodels, UML is of particular importance. UML is a well established formalism for visual modeling and has been proposed as a visual notation for knowledge representation languages as well [HEC⁺04, BKK⁺01, BKK⁺02, CP99, Kre98]. While UML itself lacks specific features of knowledge representation languages, the extension mechanisms – UML profiles – allow to tailor the visual notation to specific needs.

5. Independence from particularities of specific formalisms The metamodeling approach of MDA and MOF allows to define a model in an abstract form independent from the particularities of specific logical formalisms. This enables to be compatible with currently competing formalisms (e.g. in the case of rule- or mapping languages), for which no standard exists yet. Language mappings define the relationship with

particular formalisms and provide the semantics for the ontology model. Furthermore, the extensibility capabilities of MOF allow to add new modules to the metamodel if required in the future.

Consequences of using a metamodel Above we have discussed a number of advantages of using a metamodel-based specification. It should be noted that these are benefits that one *may* make use of, but is not obliged to.

It is possible to simply use the metamodel as a reference, but realize developments in terms of other technologies.

E.g., While MOF-based syntaxes are used in MDA environments and as such, of course other syntaxes (e.g. OWL/XML or OWL/RDF syntax in the case of OWL) may also be used. Similarly, one does not need to use a UML based syntax for visual modeling of ontologies (as for example in OntoModel¹, but one may resort to other visualizations (as e.g. for the relationship visualizer²).

3.2.3 Semantics of the Metamodel

The MOF does not completely define the semantics of the language specified by a metamodel. The semantic features of the MOF are limited to describing constraints on the structure of models, using the Object Constraint Language OCL. Obviously, this is not sufficient, and not intended, to capture the complete semantics of arbitrary languages. The actual semantics of the language specified by the metamodel is defined by so called language mappings, also called groundings. They define the relationship with particular (logical) formalisms and provide the semantics for the metamodel. For the networked ontology model this means the following:

- Our metamodel for OWL DL ontologies has a one-to-one mapping to the abstract syntax of OWL DL and thereby to its formal semantics.
- The rule metamodel currently has a translation to SWRL and its semantics. However, SWRL is not generally accepted as a standard for representing rules. Other proposals for rule languages exist, which differ in their semantics. To support such languages (such as F-Logic) in the future, it would either be possible to define new language mappings from our metamodel to these languages, or to define different rule metamodels, each of which is tailored to a specific rules language.
- For the mapping metamodel, we deliberately abstained from subscribing to a particular mapping language, but instead constructed the metamodel to be able to cover all relevant approaches to mapping languages. We believe that many of them are useful in different contexts and that a formalism-independent metamodel is appropriate. The work on groundings in specific mapping languages is work in progress and will be reported in subsequent deliverables.
- Also for the modularization metamodel, we did not subscribe to a specific language for modular ontologies. Different language mappings would allow to support various languages for modular ontologies. As part of a subsequent deliverables (D1.1.3) we will define the appropriate semantics for modular ontologies in NeOn.

¹<http://www.neon-toolkit.org/wiki/index.php/OntoModel>

²http://www.neon-toolkit.org/wiki/index.php/Plugin_for_OWL_Ontology_Visualization

Chapter 4

The OWL Metamodel

This chapter introduces the core of our networked ontology model, which is a MOF-based metamodel for OWL 2 ontologies.

We provide the metamodel for OWL 2 using the core modeling features provided by MOF. To state the metamodel more precisely, we augment it with OCL constraints, which specify invariants that have to be fulfilled by all models that instantiate the metamodel.

A metamodel for an ontology language can be derived from the modeling primitives offered by the language. Our metamodel for OWL ontologies has a one-to-one mapping to the functional-style syntax of OWL 2 and thereby to its formal semantics.¹

Along with the explanation of the various OWL constructs², we introduce and discuss the corresponding metaclasses, their properties and their constraints. To simplify the understanding of the metamodel, we add accompanying UML diagrams to our discussion.

This chapter presents the MOF-based metamodel for OWL in eight sections: Section 4.1 starts with ontologies and annotations, after which Section 4.2 presents entities and data ranges. Next, Section 4.3 demonstrates class descriptions and Section 4.4 presents OWL axioms. Then, Section 4.5 gives class axioms, after which Section 4.6 presents object property axioms. Finally, Section 4.7 presents data property axioms and Section 4.8 presents facts.

4.1 Ontologies and Annotations

An OWL ontology is defined by a set of axioms. Additionally, an ontology has an ontology IRI and an optional version IRI (which together act as a composite identifier), a (possibly empty) set of imported ontologies, and a set of annotations (see Example 1). With the import primitive, another ontology and so its axioms can be imported into an ontology. An annotation is some arbitrary information on the ontology, like the creator, version info and so forth. It is associated with an annotation property and the actual annotation value.

Example 1 *The following example illustrates the definition of an ontology and involves the specification of an imported ontology and an annotation:*

```
Ontology(PharmaceuticalDomain Import(Medication) Comment("An ontology about the pharmaceutical domain."))
```

Figure 4.1 shows the metamodel presentation for ontologies, its axioms and annotations as metaclasses. The association *axioms* connects the ontology to its axioms, whereas the associations *ontologyAnnotations*

¹Where the language specifications define constraints on character strings like for instance IRIs, this is not included as OCL constraints in the metamodel. Although it might be somehow possible to enforce such restrictions by defining OCL constraints, it would be cumbersome and thus it is preferable that tools that implement the metamodel support these constraints.

²Remember, however, that the language itself is not part of our contribution.

and *axiomAnnotations* connect ontologies respectively axioms to their annotations. The class *Ontology* has associations identify the ontology, *ontologyIRI* and *versionIRI*. Please note that a metaclass *IRI* is used to represent all IRIs as a first-class object. Finally, the association *imports* and *directlyImports* from the class *Ontology* to itself links an ontology to its (directly) imported ontologies.

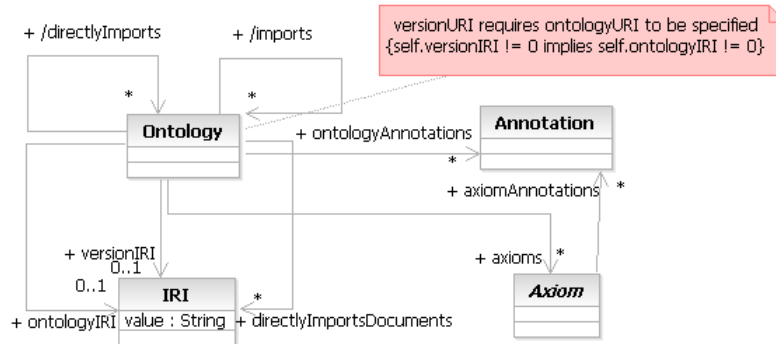


Figure 4.1: OWL metamodel: ontologies

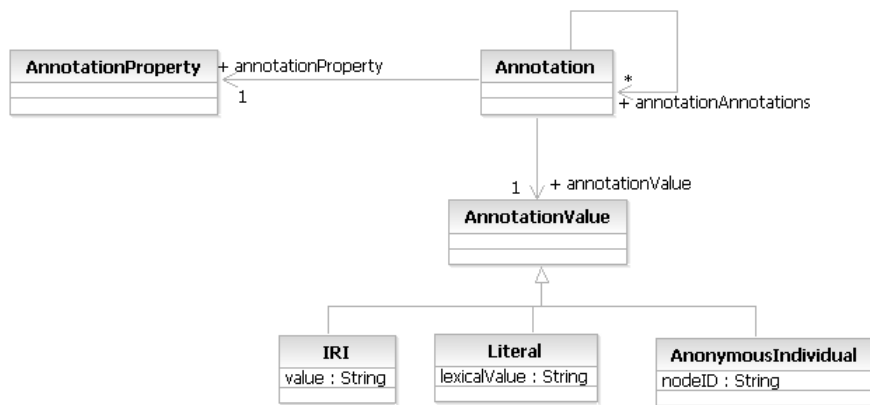


Figure 4.2: OWL metamodel: annotations

Figure 4.2 demonstrates show annotations in the metamodel. An association *annotationValue* connects an *Annotation* to its value. The OWL 2 specifications specify the value of an annotation to be an anonymous individual, an IRI or a Literal.

4.2 Entities and Data Ranges

Entities are the fundamental building blocks of OWL 2 ontologies. OWL 2 has six entity types: data types, OWL classes³ (see Example 2), named individuals, object properties, data properties, and annotation properties. A datatype is the simplest type of data range. The second entity, a class, is a simple axiomatic class description classifying a set of instances. These class instances are called individuals, which may be either named or unnamed individuals. At last, an object property connects an individual (belonging to a class) to another individual, whereas a data property connects an individual to a data value (belonging to a data range).

³OWL provides two classes with predefined IRI and semantics: owl:Thing defines the set of all objects (top concept), whereas owl:Nothing defines the empty set of objects (bottom concept).

Example 2 The following example illustrates the definition of a class in OWL:
Class(Medication)

The OWL specifications highlight entities as the main building blocks of an OWL ontology and its axioms. Hence, the metamodel defines them as first-class objects in the form of metaclasses. Figure 4.3 presents an abstract metaclass *Entity* which is defined as supertype of all types of entities. The five specific types of entities are specified as subtypes of *Entity*: *Datatype*, *Class*, *ObjectProperty*, *DataProperty* and *Individual*.

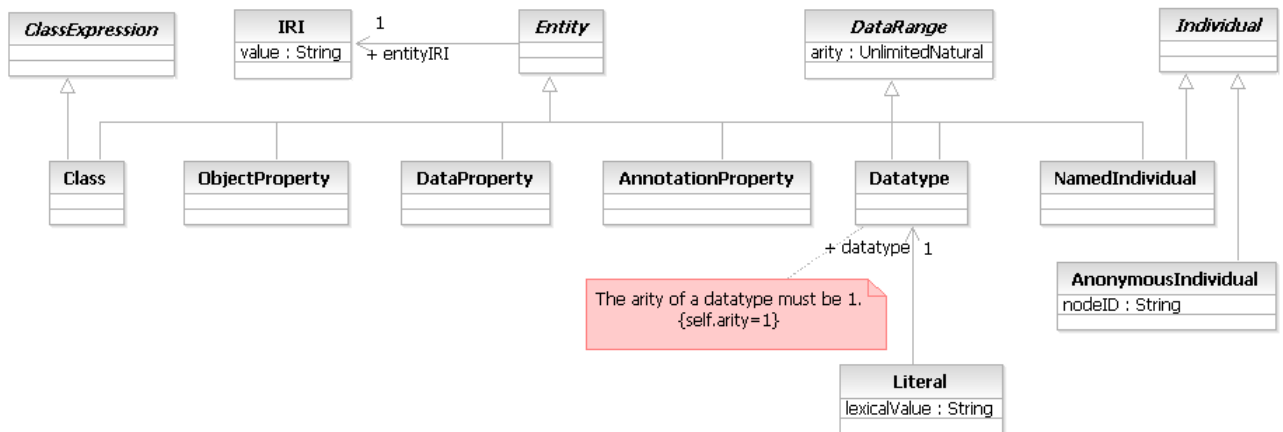


Figure 4.3: OWL metamodel: entities

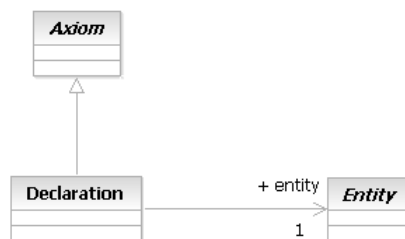


Figure 4.4: OWL metamodel: declarations

Additionally, OWL provides axioms to declare entities (see Example 3). An entity is declared in an ontology if the ontology, or one of its imported ontologies, contains a declaration axiom for the entity. Entity declarations are important for checking the structural consistency of an ontology: if each entity occurring in an axiom of the ontology is declared in the ontology, the ontology is structurally consistent. Figure 4.4 presents a metaclass *Declaration* as a subclass of the metaclass *Axiom*, linked to the entity via the association *entity*.

Example 3 The following example illustrates how a class definition is declared:
Declaration(Class(Medication))

OWL distinguishes two types of property expressions: object property expressions and data property expressions, represented by the respective metaclasses *ObjectProperty* and *DataProperty*. An object property can possibly be defined as the inverse of an existing object property (see Example 4). The metamodel has an abstract superclass *ObjectPropertyExpression* for both the usual object property and the object property defined as an inverse of another. Figure 4.5 gives its subclasses *ObjectProperty* and *InverseObjectProperty*, representing normal respectively inverse object properties. An inverse object property can be defined based on any, normal or inverse, object property, hence the association *inverseProperty* from the metaclass *InverseObjectProperty* is connected to the superclass *ObjectPropertyExpression*.

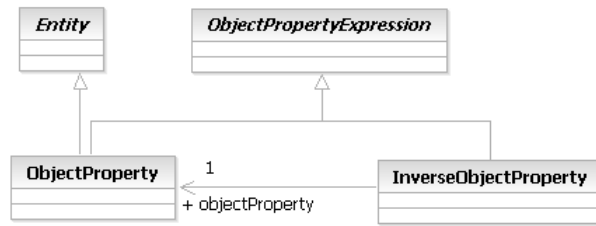


Figure 4.5: OWL metamodel: object property expressions

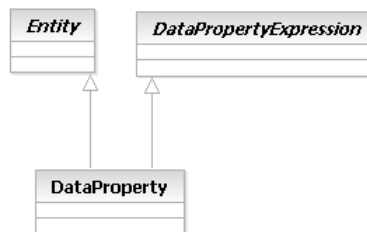


Figure 4.6: OWL metamodel: data property expressions

Example 4 The following example illustrates the definition of an inverse object property:
InverseObjectProperty(MadeFromIngredient)

Figure 4.6 shows that, although inverse data properties can not be defined, the entity *DataProperty* is also generalized into a supertype in the metamodel, in symmetry with object properties.

Figure 4.7 gives the metamodel representations for the various data range constructs in OWL. To define a range over data values, OWL provides six constructs, which are generalized in the metamodel into an abstract superclass *DataRange*. The basic and simplest data range is the datatype, defined by an IRI. The metaclass *DataComplementOf*, *DataUnionOf* and *DataIntersectionOf* define a data range as the complement, union and intersection of existing data range. The next data range construct, represented in the metamodel by the metaclass *DatatypeRestriction*, applies a facet on an existing data range, connected through the association *dataRange*. The facet is specified through the metaclass *FacetRestriction*, where the value of the facet is specified as a literal. The last data range type, represented by the metaclass *DataOneOf*, defines a data range by enumerating the data values it contains (see Example 5). The enumerated data values are represented as constants.

Example 5 The following example illustrates the enumeration of data values:
DataOneOf("5"^^xsd:integer "15"^^xsd:integer)

4.3 Class Descriptions

The two remaining groups of constructs are classes and axioms. We first address the classes. Classes group similar resources together and are the basic building blocks of class axioms. The class extension is the set of individuals belonging to the class. To define classes, OWL 2 provides next to a simple class definition (metaclass *Class*) several very expressive means for defining classes. The metamodel defines

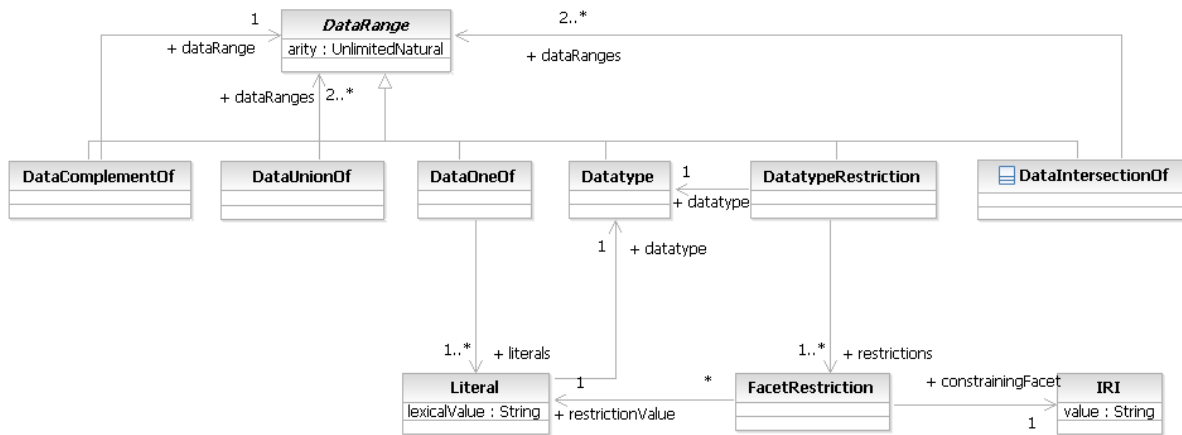


Figure 4.7: OWL metamodel: data ranges

a metaclass *Description* as abstract superclass for all class definition constructs. These constructs can be divided into two main groups: propositional connectives, and restrictions on properties. We present all class descriptions in this section, accompanied by five diagrams.

Firstly, Figure 4.8 presents the class constructs with propositional connectives⁴. Propositional connectives build classes by combining other classes or individuals. The metaclasses *ObjectUnionOf* and *ObjectIntersectionOf* represent constructs defining a class of which each individual belongs to at least one, respectively to all of the specified classes (see Example 6). They both have an association called *classExpressions* specifying the involved classes⁵. Moreover, a class can be defined as the complement of another class, represented by the metaclass *ObjectComplementOf* with association *class*, or as an enumeration of a set of (at least one) individuals. The latter one is represented in the metamodel by the metaclass *ObjectOneOf* and its association *individuals*.

Example 6 *The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description as the intersection of three other classes:*
SubClassOf(FluidMedication ObjectIntersectionOf(Fluid Medication BottledProduct))

All other class definitions in OWL 2 describe a class by placing restrictions on the class extension. Figure 4.9 gives the next group, which defines restrictions on the property value of object properties for the context of the class. The metaclass *ObjectAllValuesFrom* represents the class construct that defines a class as the set of all objects which have only objects from a certain other class description as value for a specific object property (see Example 7).

Example 7 *The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description by restricting an object property:*
SubClassOf(Medication ObjectAllValuesFrom(Treats Disease))

The OWL construct that defines a class as all objects which have at least one object from a certain class description as value for a specific object property, is represented by the metaclass *ObjectSomeValuesFrom*. Both *ObjectAllValuesFrom* and *ObjectSomeValuesFrom* have an association called *class*, specifying the class description of the construct, whereas their association *property* specifies the object property on which

⁴Note that *Class*, although shown in this figure as a subclass of *Description*, is just a simple class and no propositional connective.

⁵Note that, as the involved classes can be defined by any class description construct, they are specified in the metamodel via their abstract superclass.

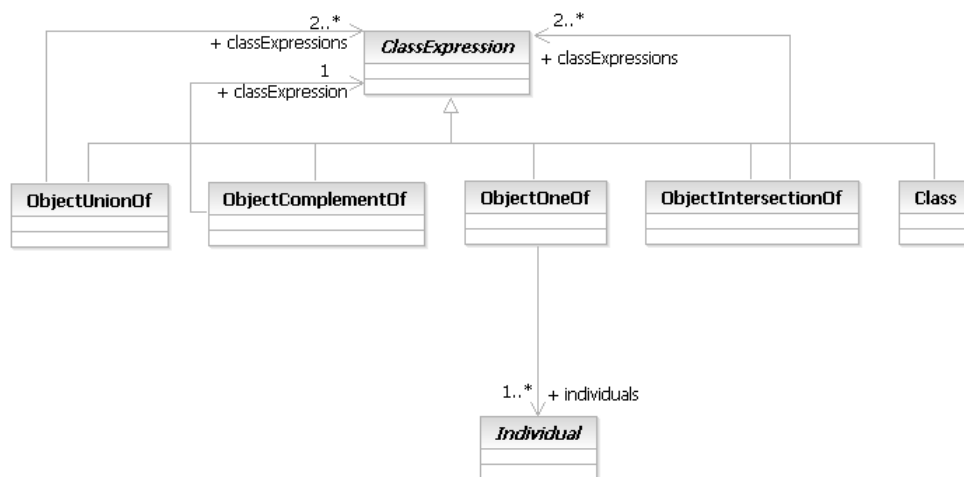


Figure 4.8: OWL metamodel: propositional connectives

the restriction is defined. To define a class as all objects which have a certain individual as value for a specific object property, OWL provides a construct that is represented in the metamodel by the metaclass *ObjectHasValue*, its association *property* specifying the object property, and its association *value* specifying the property value. The metaclass *ObjectExistsSelf* represents the class description of all objects that have themselves as value for a specific object property.

Thirdly, Figure 4.10 demonstrates the third group of class definitions, which impose restrictions on the cardinalities of object properties. A cardinality of an object property for the context of a class can be defined as a minimum, maximum or exact cardinality. The first one of this group is represented by the metaclass *ObjectMinCardinality*, which defines a class of which all individuals have at least N different individuals of a certain class as values for the specified object property (N is the value of the cardinality constraint) (see Example 8).

Example 8 *The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description by restricting the cardinality of an object property:*
SubClassOf(Medication ObjectMinCardinality(1 madeFromIngredient Ingredient))

Secondly, the construct represented by the metaclass *ObjectMaxCardinality* defines a class of which all individuals have at most N different individuals of a certain class as values for the specified object property. Finally, the construct represented by the metaclass *ObjectExactCardinality* defines a class of which all individuals have exactly N different individuals of a certain class as values for the specified object property. To specify the cardinality (N) of these constructs, which is a simple integer, all three metaclasses have an attribute *cardinality*. OCL constraints define that this cardinality must be a nonnegative integer⁶:

1. The cardinality must be nonnegative:
 context ObjectExactCardinality inv:
 self.cardinality >= 0
2. The cardinality must be nonnegative:

⁶Note that it would make sense to define a constraint which specifies that when a minimum and a maximum cardinality on a property are combined to define a class, the minimum cardinality should be less than or equal to the maximum cardinality. However, as the OWL specifications do not define this restriction and rely on OWL applications to handle this, we also do not define an OCL constraint in the metamodel.

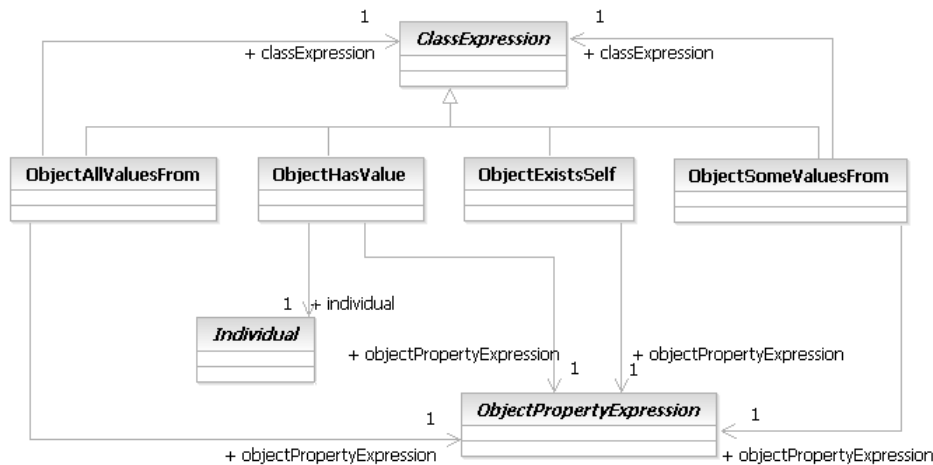


Figure 4.9: OWL metamodel: object property restrictions

context ObjectMaxCardinality inv:
self.cardinality >= 0

- The cardinality must be nonnegative:
context ObjectMinCardinality inv:
self.cardinality >= 0

Additionally, they all have an association *class* and an association *objectPropertyExpression* representing the class respectively the object property involved in the statement. Note that the multiplicity of the associations called *classExpression* have multiplicity 'zero or one' as OWL does not define the explicit specification of the restricting class description as mandatory⁷.

Just like with object properties, restrictions can be defined on data properties to define classes. On data properties only three kinds of property value restrictions can be defined (see Example 9).

Example 9 *The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description by specifying a certain value for a data property:*
SubClassOf(MedicationForAdults DataHasValue(hasMinimumAge "12"^^xsd:integer))

Figures 4.11 represents the metaclasses for these constructs, *DataSomeValuesFrom*, *DataAllValuesFrom* and *DataHasValue*. The meaning of the corresponding constructs is the same as with these restrictions on object properties. The value of a data property belongs to a data range, represented by the metaclass *DataRange*, and a specific value is represented by the metaclass *Constant*. Moreover, the associations called *properties* that both *DataAllValuesFrom* and *DataSomeValuesFrom* have, are ordered and have multiplicity '1 to many', as OWL allows to specify more than one data property. This supports class definitions like "objects whose width is greater than their height", where the width and height are specified using two data properties.

As the last set of class descriptions in OWL, Figure 4.12 gives the cardinality restrictions defined on data properties similar to the cardinality restrictions on object properties. The restrictions specify a minimum, maximum or exact cardinality on a certain data range for the specified data property. Again, OCL constraints restrict the cardinality to nonnegative integers:

⁷In the case where it is not explicitly defined, called an unqualified cardinality restriction, the description is owl:Thing.

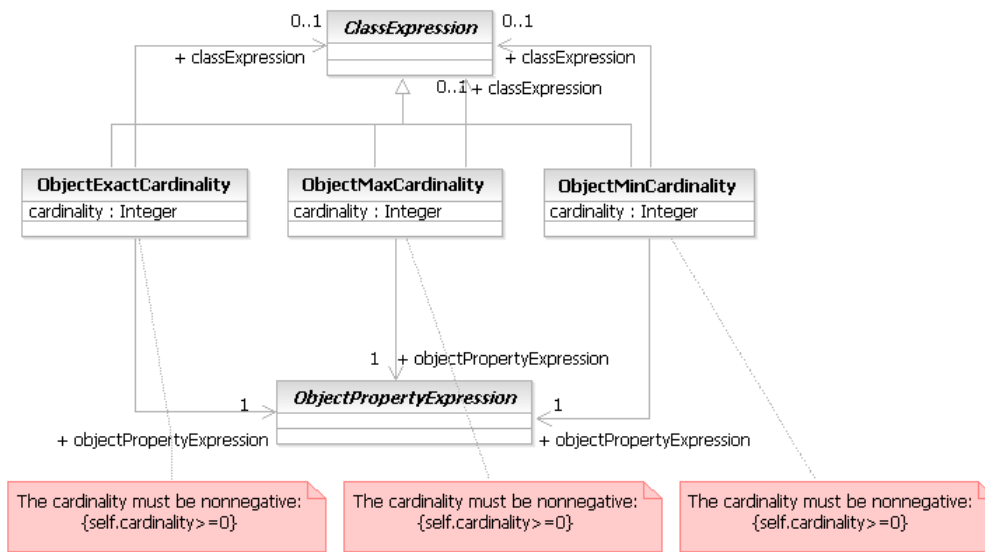


Figure 4.10: OWL metamodel: object property cardinality restrictions

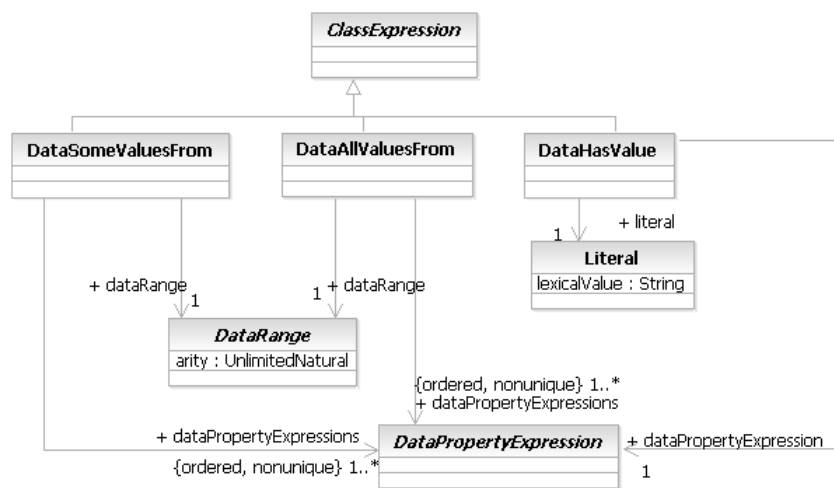


Figure 4.11: OWL metamodel: data property restrictions

1. The cardinality must be nonnegative:
context *DataMinCardinality* inv:
self.cardinality >= 0
2. The cardinality must be nonnegative:
context *DataMaxCardinality* inv:
self.cardinality >= 0
3. The cardinality must be nonnegative:
context *DataExactCardinality* inv:
self.cardinality >= 0

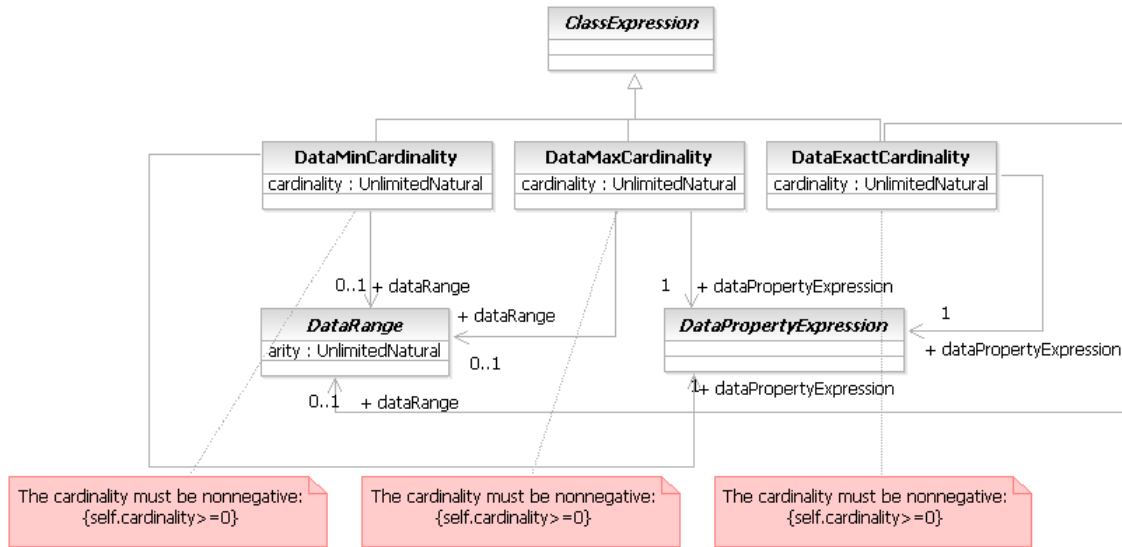


Figure 4.12: OWL metamodel: data property cardinality restrictions

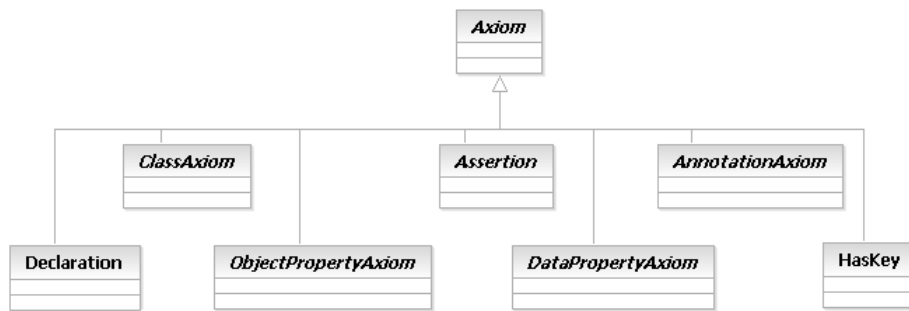


Figure 4.13: OWL metamodel: axioms

4.4 OWL Axioms

Figure 4.13 demonstrates the seven types of OWL axioms. We now introduce the remaining types of axioms, which are class axioms, object property axioms, data property axioms and assertions and keys. They are all defined through an abstract superclass with various subclasses⁸.

4.5 Class Axioms

Class axioms make statements about the extensions of two or more classes. Figure 4.14 gives the metamodel representations for the four kinds of OWL class axioms. The first class axiom defines that a class is a subclass of another class and is represented in the metamodel as the metaclass *SubClassOf*, connected to the two classes via the associations *subClassExpression* and *superClassExpression*. Two other axioms, represented by the metaclasses *DisjointClasses EquivalentClasses* and *DisjointUnionClasses*, define that the extensions of two or more classes are disjoint, respectively equivalent (see Example 10). An association

⁸Note that the metamodel models all these statements as metaclasses since OWL specifies them as first-class objects in the form of axioms.

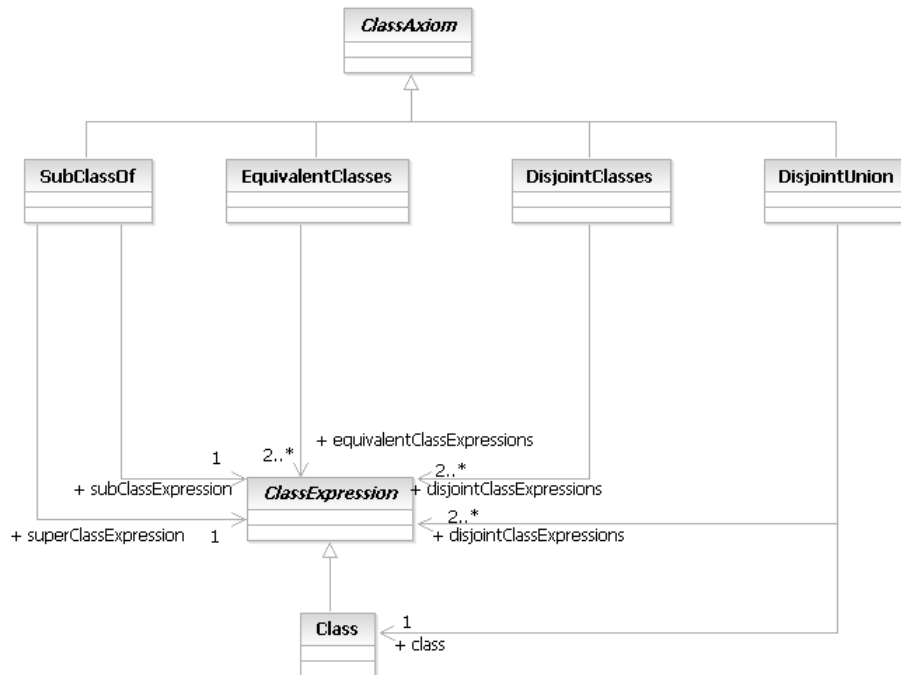


Figure 4.14: OWL metamodel: class axioms

disjointClassExpressions, respectively *equivalentClassExpressions* connects the metaclasses to the related class descriptions.

Example 10 The following example illustrates how three classes are defined to be equivalent: *EquivalentClasses(Disease Illness Sickness)*

The fourth and last class axiom defines that one class is the union of a set of classes which are all pair-wise disjoint. The metaclass *DisjointUnion* represents this axiom and has an association *unionClassExpressions* specifying the class, and an association *disjointClassExpressions* specifying the disjoint classes.

4.6 Object Property Axioms

The group of object property axioms contains constructs defining relations between different properties, definitions of domain and range and specifications of property characteristics. We first introduce the various relations between object properties in Figure 4.15.

The first one, represented by the metaclass *SubObjectPropertyOf*, defines that the property extension of one object property, specified through the association *subObjectPropertyExpression*, is a subset of the extension of another object property, specified through the association *superObjectPropertyExpression*. OWL allows to specify a subproperty chain, where the relation defined through the application of the first until the last property of the chain is defined as the subproperty of the specified superproperty. Hence the association *subObjectPropertyExpressions* is ordered and carries multiplicity 'one to many'. The metaclass *EquivalentObjectProperties* represents a construct that defines that the property extensions of two or more object properties are the same, whereas the class *DisjointObjectProperties* connects two or more object properties that are pair-wise disjoint. Inverse object properties are combined with the construct represented as the metaclass *InverseObjectProperties*, which defines that for every (x,y) in the property extension of one property, there is a pair (y,x) in the other property extension, and vice versa.

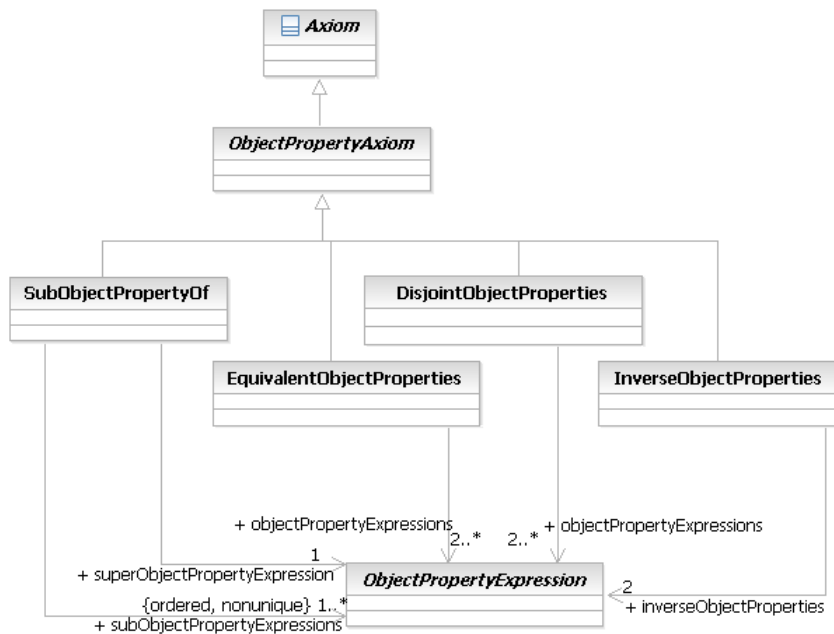


Figure 4.15: OWL metamodel: object property axioms - part 1

To define the class to which the subjects of an object property belong, OWL provides the domain concept (see Example 11), whereas a range specifies the class to which the objects of the property, the property values, belong. Figure 4.16 presents the metamodel elements for the constructs defining an object property domain and range.

Example 11 *The following example illustrates the definition of the domain of an object property:*
ObjectPropertyDomain(madeFromIngredient Medication)

The metaclass *ObjectPropertyDomain* specifies an object property and its domain via the associations *property* respectively *domain*. Similarly, the metaclass *ObjectPropertyRange* represents the construct to define the range of an object property.

The remaining OWL object property axioms take an object property and assert a characteristic to it. In doing so, object properties can be defined to be functional, inverse functional, reflexive, irreflexive, symmetric, antisymmetric, or transitive.

A functional object property is a property for which each subject from the domain, can have only one value in the range, whereas an inverse functional object property can have only one subject in the domain for each value in the range. A transitive property defines that when the subject-object pairs (x,y) and (y,z) belong to the property extension, then the pair (x,z) belongs to the property extension as well.

When an object property is defined to be reflexive, then for each object x, the subject-object pair (x,x) belongs to the object property extension. The opposite, when for each object x, the subject-object pair (x,x) does not belong to the object property extension, can be specified as an irreflexive object property. When an object property is specified to be symmetric or asymmetric, then the pair (y,x) does respectively does not belong to the object property extension when the subject-object pair (x,y) belongs to the object property extension (see Example 12).

Example 12 *The following example illustrates how an object property is defined to be asymmetric:*
AsymmetricObjectProperty(madeFromIngredient)

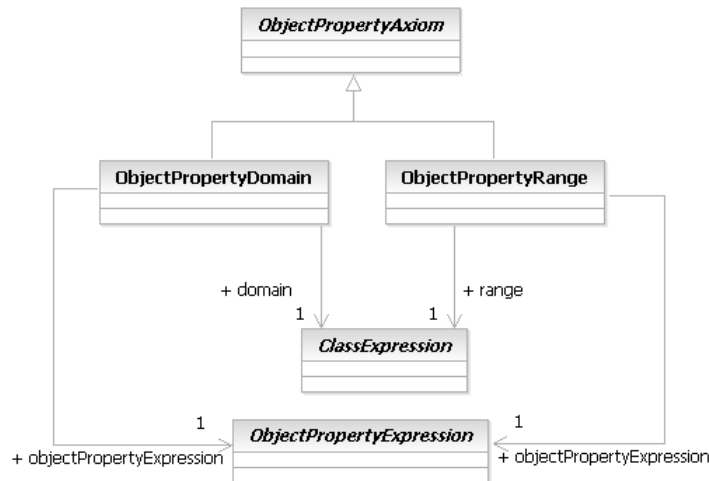


Figure 4.16: OWL metamodel: object property axioms - part 2

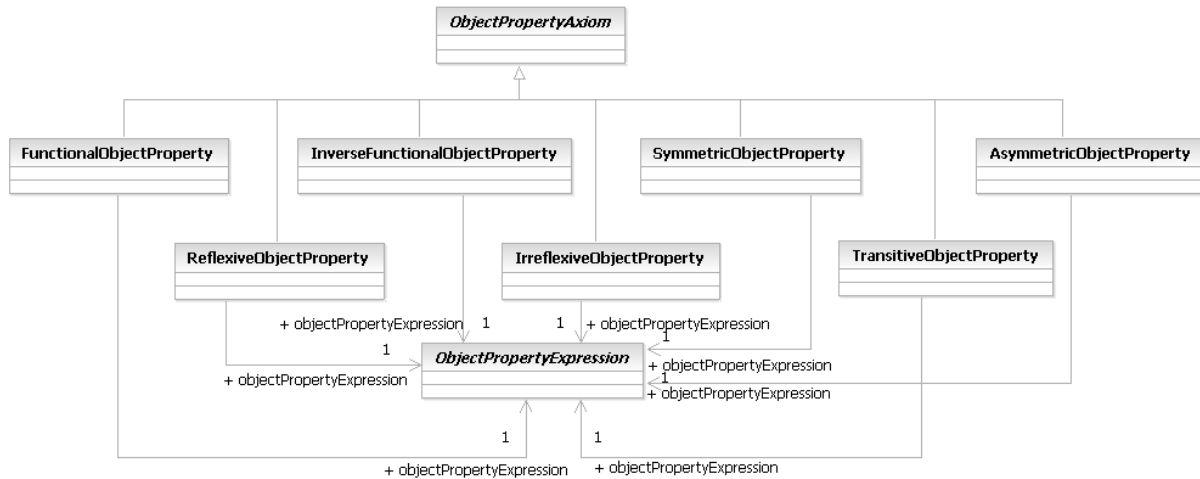


Figure 4.17: OWL metamodel: object property axioms - part 3

Figure 4.17 demonstrates that each of these axioms has an own metaclass with an association *objectPropertyExpression* to the class *ObjectPropertyExpression*, specifying the property on which the characteristic is defined.

4.7 Data Property Axioms

Similar to object property axioms, OWL provides six types of axioms on data properties.

Data properties can be defined to be a subproperty of another data property, or can be defined as being functional. Additionally, two or more data properties can be defined to be equivalent or disjoint. Figure 4.18 presents the corresponding metamodel elements.

Finally, OWL provides constructs to define the domain and range of a data property (see Example 13).

Example 13 *The following example illustrates the specification of the range of a data property:*
DataObjectPropertyRange(hasMinimumAge xsd:nonNegativeInteger)

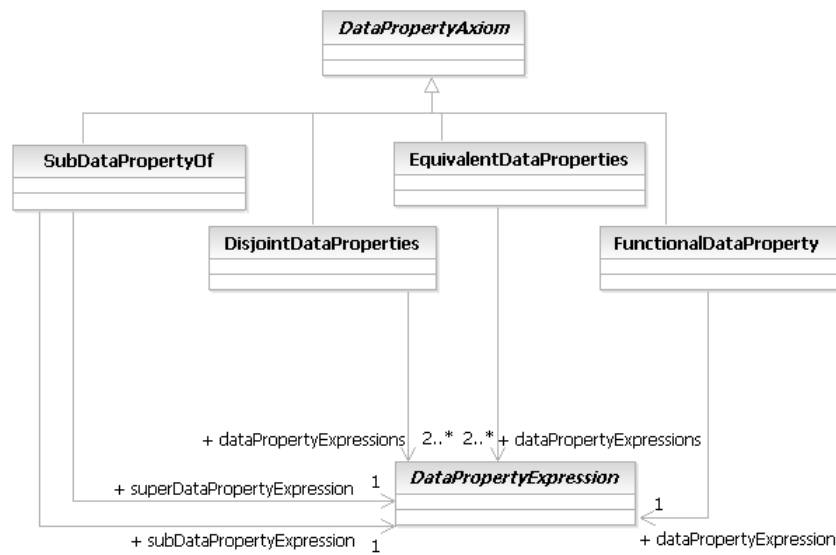


Figure 4.18: OWL metamodel: data property axioms - part 1

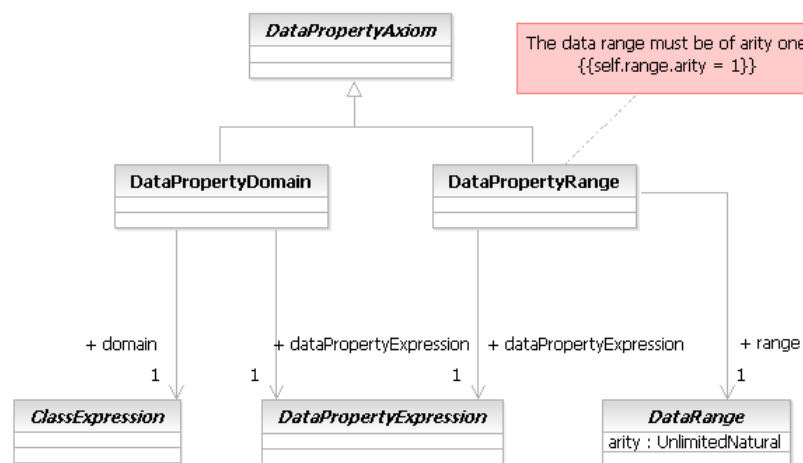


Figure 4.19: OWL metamodel: data property axioms - part 2

Figure 4.19 gives the classes that represent them in the metamodel, *DataPropertyDomain* and *DataPropertyRange*. Both have an association *dataPropertyExpression* specifying the property, and additionally an association *domain* to *Description* specifying its domain, respectively an association *range* to *DataRange* specifying its range.

4.8 Assertions

Seven different axioms allow to state facts in ontologies. Figure 4.20 gives the metamodel representations for the first three of these axioms, stating facts about classes and individuals. Since OWL is made for use on the web and one can not assume that everyone uses the same name for the same thing on the web, OWL does not have the so-called unique name assumption. Instead, OWL provides several constructs to define

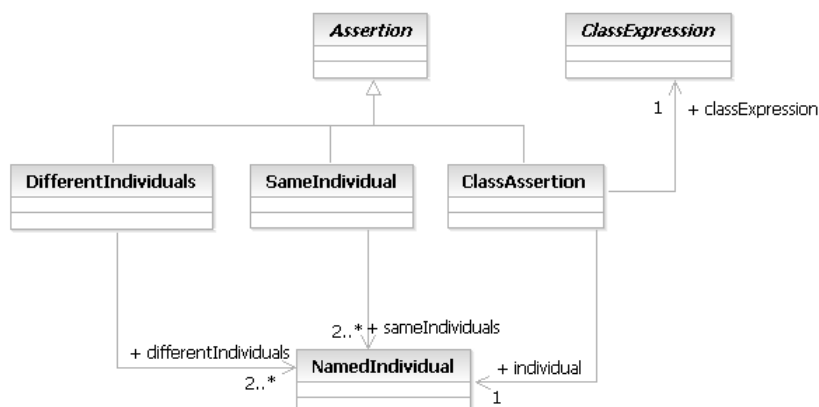


Figure 4.20: OWL metamodel: Assertions - part 1

facts about the identity of individuals: individuals can be defined to denote the same object, represented in the metamodel by the metaclass *SameIndividual*, and two or more individuals can be defined to denote different objects, represented by the class *DifferentIndividuals*. Both metaclasses have an association to the metaclass *NamedIndividual* connecting the axiom with the individuals it applies. Moreover, a class assertion specifies to which class a certain individual belongs (see Example 14) and is defined in the metamodel as the metaclass *ClassAssertion* with an association *individual* to *Individual* and an association *class* to *Description*.

Example 14 *The following example illustrates how the class to which an individual belongs, is specified: `ClassAssertion(sinusitis Disease)`*

The remaining assertion axioms state assertions about properties. To specify the value of a specified individual under a certain object property, OWL provides the object property assertion construct (see Example 15).

Example 15 *The following example illustrates how the value of an individual under an object property, is defined:*

`ObjectPropertyAssertion(madeFromIngredient aspirin acetylsalicylicacid)`

The opposite is defined by a negative object property assertion, which defines that an individual is exactly not the value of another individual under the specified object property. Figure 4.21 shows that both constructs are represented in the metamodel as a metaclass, *ObjectPropertyAssertion* respectively *NegativeObjectPropertyAssertion*. Both have an association to *objectPropertyAssertion* representing the involved property, and two associations to *Individual* representing the subject and the object of the assertion.

Finally, as the last constructs for assertions in OWL, Figure 4.22 introduces instantiations of data properties. Similar to object property instantiations, OWL allows normal (positive) as well as negative data property assertions. Both metaclasses *DataPropertyAssertion* and *NegativeDataPropertyAssertion* have three associations connecting them to the property, an individual as the subject of the property assertion, and a constant as the property value.

4.9 Annotation Axioms

The final type of axioms are annotation axioms, as shown in Figure 4.23. These include axioms to define the properties of annotation properties (analogous to data- and object property axioms) and the actual annotation assertions. These axioms have no effect on the direct semantics of OWL 2.

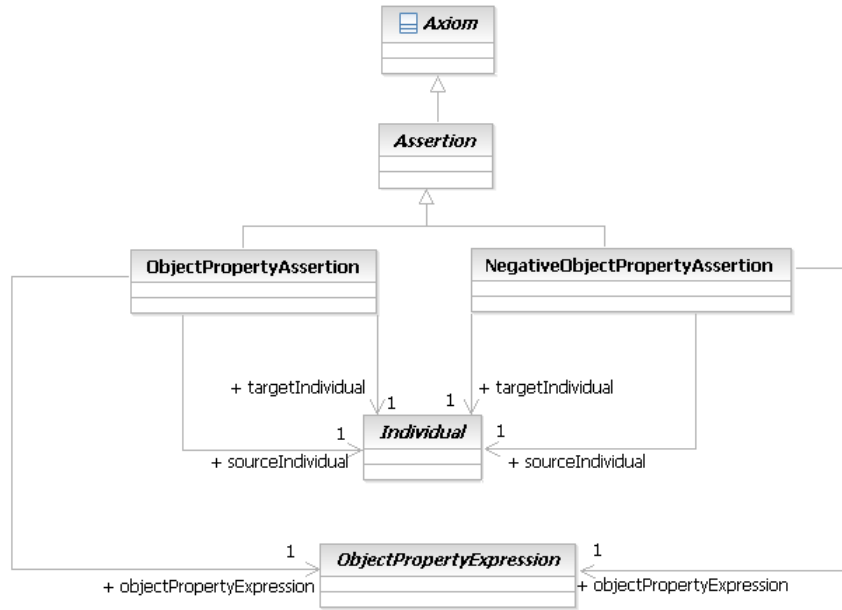


Figure 4.21: OWL metamodel: Assertions - part 2

An annotation assertion states that the annotation subject \hat{U} an IRI or an anonymous individual \hat{U} is annotated with an annotation property and an annotation value. An annotation subproperty axiom states that one annotation property is a subproperty another. An annotation property domain (range) axiom states that what the domain (range) of the annotation property is.

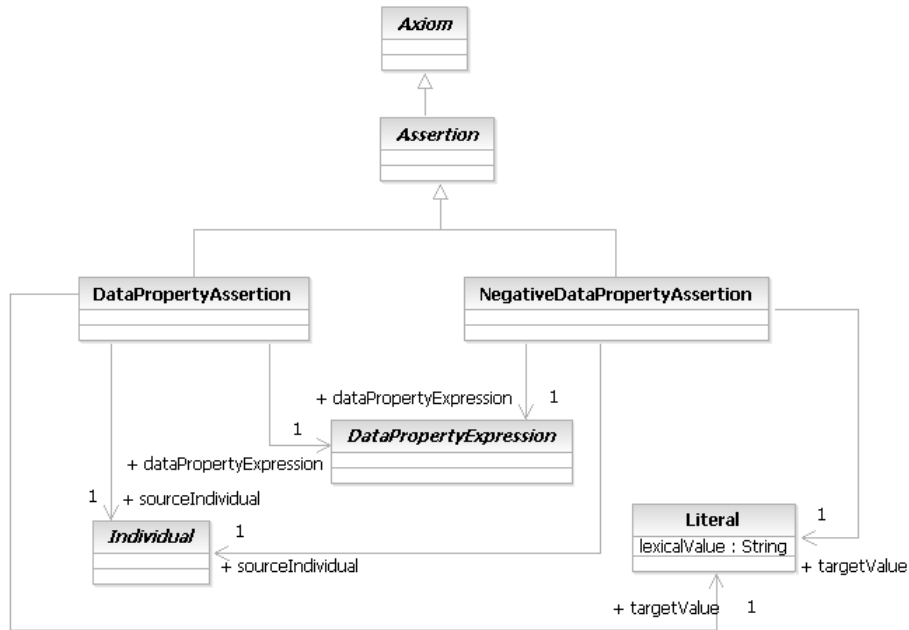


Figure 4.22: OWL metamodel: Assertions - part 3

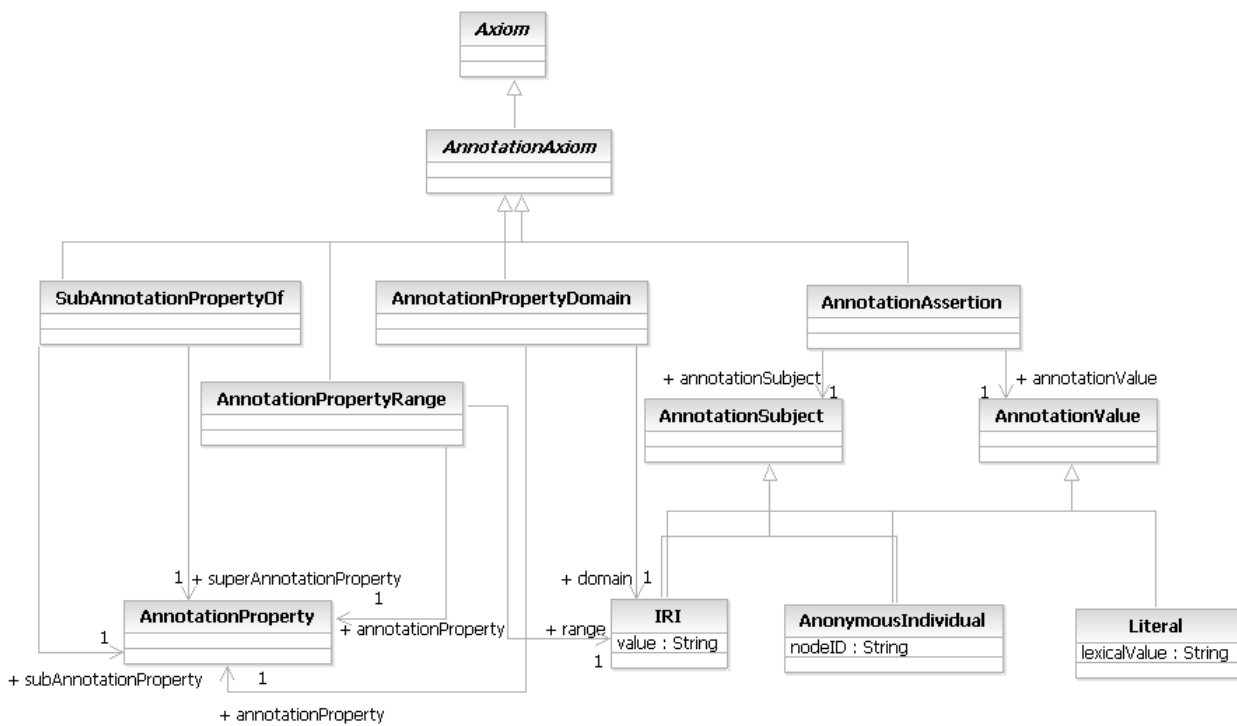


Figure 4.23: Annotation Axioms

Chapter 5

The Rule Metamodel

5.1 A Metamodel for SWRL Rules

We consistently extend our metamodel for OWL 2 presented in the previous section with a metamodel for SWRL that directly resembles the extensions of SWRL to OWL. Just like the OWL metamodel, this extension is augmented with OCL constraints to state the metamodel more precisely.

Three subsections introduce the different parts of the metamodel extension along with a discussion of the corresponding SWRL constructs.^{1 2}

The following subsections present the extension of the OWL metamodel for SWRL. Section 5.1.1 starts with rules, after which Section 5.1.2 presents predicate symbols. Finally, Section 5.1.3 presents terms.

5.1.1 Rules

The main component of OWL ontologies is the set of axioms. SWRL specifies a rule also as an axiom and Figure 5.1 demonstrates how our metamodel consequently represents it as a subclass of the abstract superclass *OWLAxiom*, called *Rule*. As all OWL axioms, a SWRL rule can be annotated, which is represented through an association of its superclass (Figure 4.1 on page 31). To identify a rule and assure compatibility with OWL, they can be assigned a URI, represented by the optional attribute *URI*.

A rule in SWRL contains an antecedent, also referred to as 'body', and a consequent, also referred to as 'head'. As we wanted to be able to treat them as first-class objects in our metamodel, we represent them by metaclasses, called *Antecedent* and *Consequent*. Both antecedent and consequent contain a number of atoms, possibly zero, and multiple atoms are treated as a conjunction in SWRL. Consequently, a rule actually says that *if* all atoms in the antecedent hold, *then* the consequent holds³. The fact that antecedent and consequent are possibly empty, is reflected in the metamodel by the multiplicity 'zero to many' on the association *bodyAtoms* from *Antecedent* to *Atom* and the association *headAtoms* between *Consequent* and *Atom*.

A SWRL atom is composed of a predicate symbol followed by an ordered set of terms. The connections between the atom and these atom components are represented in the metamodel through the associations from the metaclass *Atom* to the metaclasses *PredicateSymbol* and *Term*. Both *PredicateSymbol* and *Term*

¹Classes or relationships that already exist in the OWL metamodel, can be recognized in the accompanying UML diagrams by colored elements and an appropriate little icon.

²Remember that the language SWRL itself does not belong to our contribution.

³An empty antecedent is treated as trivially true, i.e. satisfied by every interpretation, whereas an empty consequent is treated as false, i.e. not satisfied by any interpretation.

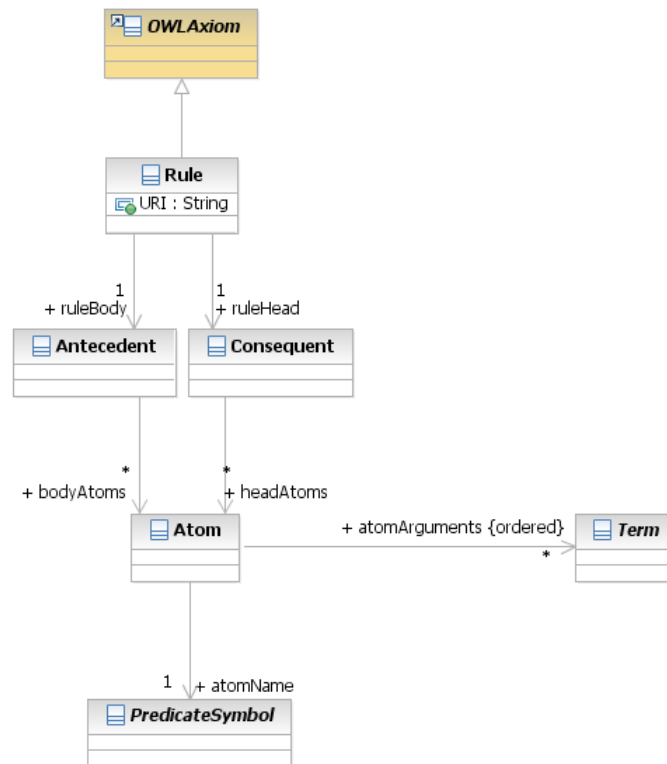


Figure 5.1: SWRL metamodel extension: rules

are abstract classes. Most of their subclasses exist already in the OWL metamodel.

5.1.2 Predicate Symbols

An atom in SWRL rules can have the following forms:

- An OWL class description, defined on a variable or an OWL individual. The atom holds if the value of the variable or individual belongs to the class description.
- An OWL data range specification using a variable or a data value. The atom holds if the variable or individual belongs to the data range.
- An OWL property. In case of an object property, the subject and object of the property are both an individual or a variable. If the specified property is a datatype property, the atom takes an individual or a variable as the subject, and a variable or data value as the object. The atom holds if the object of the property is related to the subject by the specified property.
- A 'sameAs' construct, defined on two objects which are both an individual or a variable. This construct is actually just some syntactic sugar and could be represented using other existing constructs. However, since it is useful in practice, the SWRL specifications define it as one of the basic constructs. The atom holds if the two terms are the same.
- A 'differentFrom' construct, defined on two objects which are both an individual or a variable. Just as well as *sameAs*, also *differentFrom* is syntactic sugar but very practical. The atom holds if the two terms are different.
- A built-in construct with a built-in ID and a set of variables and data values. Built-ins are classified into seven modules, like built-ins for lists or built-ins for comparisons. Implementations could select the

modules to be supported. The atom holds if the relation defined through the built-in ID, holds for the specified terms.

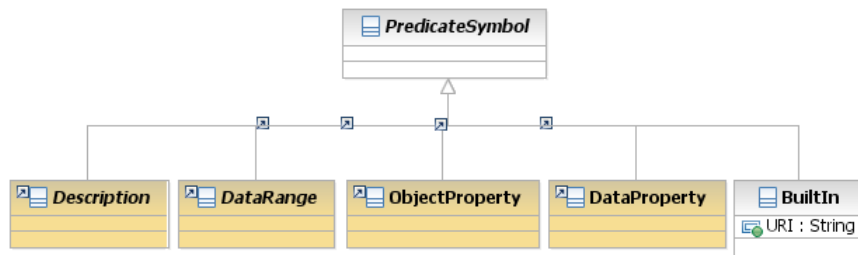


Figure 5.2: SWRL metamodel extension: predicate symbols

Figure 5.2 shows that almost all predicate are available in the OWL metamodel. The set of existing predicate symbols is extended in SWRL with one new predicate type, built-ins⁴. These built-ins are represented by the metaclass *BuiltIn* which has an attribute *URI* for their identification.

5.1.3 Terms

The last component in SWRL rules, is the set of terms. Figure 5.3 presents the metamodel elements for the possible terms in rule atoms: variables, constants and individuals.

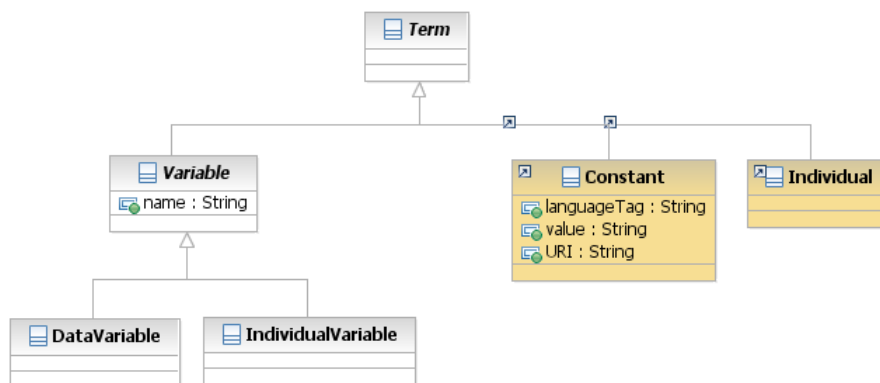


Figure 5.3: SWRL metamodel extension: terms

Variables do not exist in OWL, so they are newly introduced in the SWRL extension of the metamodel. Similar to the representation of data values and individuals, variables are also represented by classes. As two types of variables are allowed, an abstract superclass *Variable* is defined and both types of variables are defined by the subclasses *DataVariable* and *IndividualVariable*. The name of any variable is defined through the attribute *name* of the superclass, and their scope is always limited to the rule itself. Depending on the predicate symbol of the atom, only a certain number of terms as well as certain types of terms are allowed. These restrictions as well as the fact that variables that occur in the consequent must occur in the antecedent, are defined on the metamodel as OCL constraints:

1. When the predicate symbol of the atom is a description, the atom has exactly one argument and this argument must be an individual variable or an individual:

context Atom inv:

self.atomName.ocIsTypeOf(Description) implies

⁴Note that the predicates *sameAs* and *differentFrom* are represented through the metaclass *BuiltIn*.


```
self.atomArguments→size()=1 and
(self.atomArguments→at(1).oclIsTypeOf(Individual) or
self.atomArguments→at(1).oclIsTypeOf(IndividualVariable))
```

2. When the predicate symbol of the atom is a datarange, the atom has exactly one argument and this argument must be a constant or a data variable:

```
context Atom inv:
self.atomName.oclIsTypeOf(DataRange) implies
self.atomArguments→size()=1 and
(self.atomArguments→at(1).oclIsTypeOf(Constant) or
self.atomArguments→at(1).oclIsTypeOf(DataVariable))
```

3. When the predicate symbol of the atom is an object property, the atom has exactly two arguments and each of these is either an individual or an individual variable:

```
context Atom inv:
self.atomName.oclIsTypeOf(ObjectProperty) implies
self.atomArguments→size()=2 and
self.atomArguments→forAll(oclIsTypeOf(Individual) or
oclIsTypeOf(IndividualVariable))
```

4. When the predicate symbol of the atom is a data property, the atom has exactly two arguments and the first argument must be an individual or an individual variable, and the second argument must be a constant or a data variable:

```
context Atom inv:
self.atomName.oclIsTypeOf(DataProperty) implies
self.atomArguments→size()=2 and
(self.atomArguments→at(1).oclIsTypeOf(Individual) or
self.atomArguments→at(1).oclIsTypeOf(IndividualVariable)) and
(self.atomArguments→at(2).oclIsTypeOf(Constant) or
self.atomArguments→at(2).oclIsTypeOf(DataVariable))
```

5. When the predicate symbol of the atom is a built-in, the atom can have zero to many arguments and all these arguments can be either a constant or a data variable:

```
context Atom inv:
self.atomName.oclIsTypeOf(BuiltIn) implies
self.atomArguments→forAll(oclIsTypeOf(Constant) or oclIsTypeOf(DataVariable))
```

6. Only variables that occur in the antecedent may occur in the consequent of the rule:

```
context Rule inv:
self.ruleHead.headAtoms→atomArguments→forAll(t | t.oclIsTypeOf(Variable) implies
self.ruleBody.bodyAtoms→atomArguments→exists(t | true))
```

5.2 A Metamodel for F-Logic Rules

This chapter presents a metamodel for the language F-Logic. The metamodel for F-Logic is not related to the metamodel for OWL and SWRL in the sense that it is an extension, but it is a stand-alone MOF-based metamodel for a different language. The only relation that exists between the two is the fact that they are both defined in MOF, which allows to use the automatic transformation possibilities of the MDA framework. The F-Logic metamodel also contains OCL constraints to state it more precisely.

The discussion of the F-Logic metamodel starts with introducing the various metaclasses with their properties and constraints, while explaining the F-Logic constructs they represent⁵. UML diagrams accompany this discussion for the sake of clarity.

Eight subsections present the metamodel: Section 5.2.1 starts with F-Logic programs, after which Section 5.2.2 presents terms. Next, Section 5.2.3 presents Formulas and Section 5.2.4 presents rules and queries. Then, Section 5.2.5 gives logical connectives and Section 5.2.6 presents logical quantifiers. Finally, Section 5.2.7 demonstrates F-atoms and F-molecules, whereas Section 5.2.8 P-atoms.

5.2.1 F-Logic Programs

The F-Logic specifications define an F-Logic program as a set of facts and rules. Facts are expressions that represent statements about objects and their relationships. Queries take the form of rules without heads and operate on a given F-Logic program. Typically, F-Logic applications group the facts, rules and queries together and call it an F-Logic ontology.

Figure 5.4 describes the core of our metamodel and defines a metaclass *FLogicOntology* connected to a set of *Formulas* via the association *ontologyFormulas*. The class *Formula* is an abstract superclass of classes representing rules, facts and queries.

Formula also has subclasses that do not represent facts, rules or queries but are partial formulas that are contained in rules and queries. Due to the potential overlap between formulas that are contained directly in an ontology, and the set of partial formulas indirectly contained in an ontology, the metamodel generalizes both groups into one superclass.

OCL constraints specify which of the subclasses can be directly contained in *FLogicOntology*.

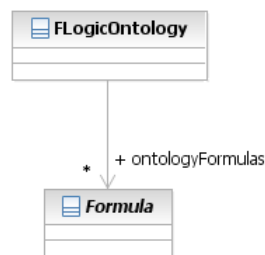


Figure 5.4: F-Logic metamodel: ontologies

5.2.2 Terms

The basic syntactical elements of F-Logic are predicate symbols, function symbols, and variables, where functional terms and variables are called id-terms in F-Logic, which identify objects, methods and classes. Functional terms are id-terms that have other id-terms as arguments. The F-Logic specifications define constants as functional terms with zero arguments. To distinguish between constants and variables, every variable has to be bound to a logical quantifier.

Figure 5.5 depicts how the various types of F-Logic id-terms are derived from an abstract metaclass called *Term*, carrying an attribute *name* that is inherited by all subclasses. Variables and functional terms are represented by the classes *Variable* respectively *FunctionalTerm*. An ordered association *arguments* to the class *Term* specify its arguments in the right order. The association carries a multiplicity of 'zero to many',

⁵Remember, however, that we did not contribute to the language F-Logic itself.

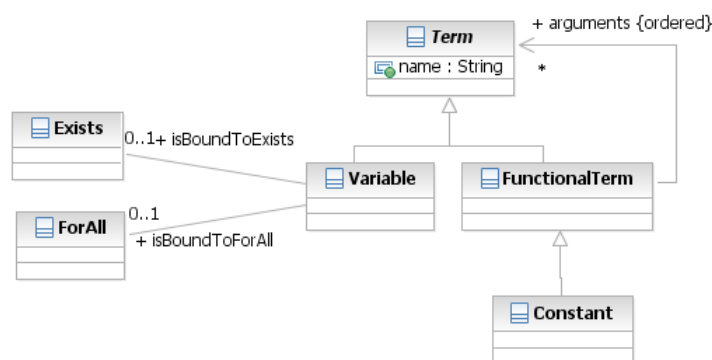


Figure 5.5: F-Logic metamodel: terms

denoted by a star, as the number of arguments in a functional term is unrestricted. For the specific type of functional terms that have zero arguments, constants, the metamodel has a specific class as it is a very common id-term. Two OCL constraints restrict the multiplicity of the association *arguments* for constants as well as for functional terms that are not a constant:

1. A constant is a functional term with zero arguments:
context Constant inv:
self.arguments→size()=0
2. A functional term that is not a constant must have at least one argument:
context FunctionalTerm inv:
self→forAll(not oclIsTypeOf(Constant) implies self.arguments→size()>0)

Finally, the metaclasses *Exists* and *ForAll* are connected to the class *Variable* via respective associations *isBoundToExists* and *isBoundToForAll* to connect a variable to the logical quantifier it is bound to.

Although every instantiation of *Variable* has to be connected to an instance of one of the two classes, the associations have multiplicity 'one or zero' as two multiplicities 'exactly one' would define that every variable needs to be connected to both classes. An OCL constraint makes sure that an instantiated variable does always have one connection to any of the two:

1. A variable must be bound to exactly one universal quantifier or existential quantifier:
context Variable inv:
(self.isBoundToExists=1 or self.isBoundToForAll=1)
and
(self.isBoundToForAll=1 implies self.isBoundToForExists=0)

5.2.3 Formulas

Facts, rules and queries are represented in the metamodel by subclasses of the metaclass *Formula*, which is linked to *FLogicOntology* through an association. Figure 5.6 gives an overview of the partial formulas that are derived from the class *Formula*. We distinguish four groups of partial formulas:

- Logical connectives: classes *Conjunction*, *Negation*, *Equivalence*, *Disjunction* and *Implication*
- Logical quantifiers: classes *ForAll* and *Exists*
- Facts: classes *PAtom*, *FMolecule* and *FAtom*

- Rules and queries: classes *Rule* and *Query*

An OCL constraint restricts the metamodel so that the logical connectives and quantifiers can not be contained directly into an ontology:

1. The only subtypes of the class *Formula* that can be directly in an ontology, are F-molecules (and so F-atoms), P-atoms, rules and queries:

context FLogicOntology inv:

self.ontologyFormulas→forall(oclIsTypeOf(FMolecule) or oclIsTypeOf(Rule) or oclIsTypeOf(Query) or oclIsTypeOf(PAtom))

The next sections explain the details of the different subclasses of *Formula*, and explains which combinations are possible.

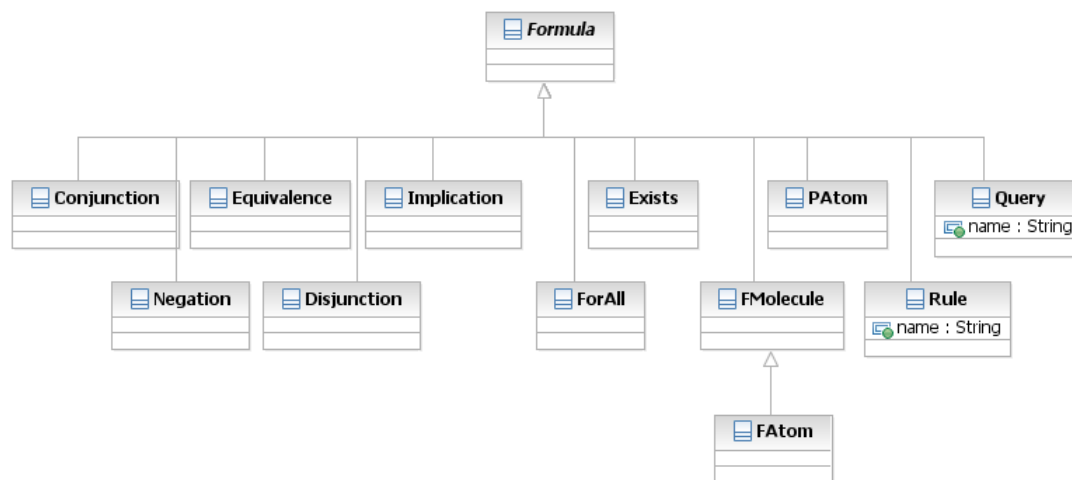


Figure 5.6: F-Logic metamodel: formulas

5.2.4 Rules and Queries

Rules (see Example 16) infer new information from available facts and so extend the object base intensionally. A rule consists of a head (postcondition), an implication sign (' \leftarrow '), and a body (precondition). A rule defines that *when* the formula (or combination of formulas) in the body is satisfied, *then* the formula (or combination of formulas) in the head is satisfied. The rule head is a conjunction of P-atoms, F-Molecules and F-atoms, all connected through 'AND'. In the body, arbitrary formulas are allowed, and P-atoms, F-atoms and F-molecules can be connected by any of the logical connectives: implies (' \rightarrow '), implied by (' \leftarrow '), equivalent (' \leftrightarrow '), and ('AND'), or ('OR'), and not ('NOT').

Example 16 *A pharmacy that is a client of a certain pharmaceutical lab, can sell the medications produced by that lab:*

$$\forall X, Y, Z \ X : Pharmacy[canSell \rightarrow Z] \leftarrow$$

$$X : Pharmacy[clientOf \rightarrow Y] \wedge Y : Laboratory[produces \rightarrow Z].$$

When a rule contains variables, they must be introduced using a quantifier. A universal quantifier (' \forall ') can appear in front of a rule or in the rule body, and an existential quantifier (' \exists ') can also appear anywhere in the rule body.

From the rules and facts in an F-Logic program, a model is computed on which queries (see Example 17) can be asked. The result of a query is a set of substitutions for the query's variables, that can be derived from the facts and rules in the knowledge base. Note that also schema level queries are allowed, where not only instances and their values but also concept and attribute names can be provided as answers via variable substitutions.

Example 17 The following example query retrieves sets of variable substitutions 'medication, person, disease' whereas the medication is made from the ingredient 'acetylsalicylicacid', and the person has a disease against which the medication helps. The following values are a possible result of the query: $X = \text{Aspirin}$, $Y = \text{APerson}$, $Z = \text{headache}$:

$$\forall X, Y, Z \leftarrow X : \text{Medication}[\text{madeFromIngredient} \rightarrow \text{acetylsalicylicacid}, \text{helps} \rightarrow Y] \wedge Y[\text{hasDisease} \rightarrow Z]$$

Figure 5.7 depicts the representation of rules and queries as the metaclasses *Rule* respectively *Query* in the metamodel. As the F-Logic specifications define that rules and queries can be given a name, both classes contains the attribute *name*.

Both *Rule* and *Query* are connected to the formula it contains through an association *containedFormula* carrying multiplicity 'one'⁶.

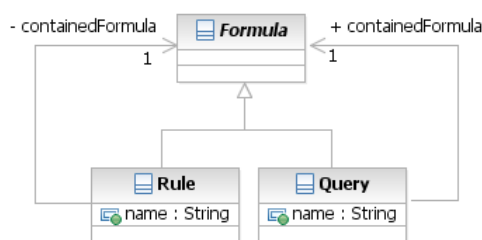


Figure 5.7: F-Logic metamodel: rules and queries

Several OCL constraints define how partial formulas can be combined in a rule:

1. A rule is an implication formula and possibly has a universal quantifier in front:

context Rule inv:

(self.containedFormula.oclsTypeOf(Implication) or

self.containedFormula.oclsTypeOf(ForAll))

and

(self.containedFormula.oclsTypeOf(ForAll) implies

self.containedFormula.oclAsType(ForAll).containedFormula.oclsTypeOf(Implication))

2. The head of a rule is a conjunction of facts or is just one fact (F-molecule or P-atom)⁷. The following constraint defines this for rules without a universal quantifier:

context Rule inv:

self.containedFormula.oclsTypeOf(Implication) implies

self.containedFormula.oclAsType(Implication).consequent.oclsTypeOf(Conjunction) or

self.containedFormula.oclAsType(Implication).consequent.oclsTypeOf(FMolecule) or

self.containedFormula.oclAsType(Implication).consequent.oclsTypeOf(PAtom)

⁶Remember that when a rule or query consist of a combination of more than one formula, this is also represented as one formula, hence they can always contain maximum one formula.

⁷Note that another fact construct in F-Logic, F-atom, is defined as a subclass of *FMolecule* and thus is also included.

3. A similar constraint is defined for rules with universal quantifier:
context Rule inv:
self.containedFormula.ocllsTypeOf(ForAll) implies
self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
consequent.ocllsTypeOf(Conjunction) or
self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
consequent.ocllsTypeOf(FMolecule) or
self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
consequent.ocllsTypeOf(PAtom)

4. When the head of a rule is a conjunction, the combined formulas may only be facts. The following constraint defines this for rules without a universal quantifier:
context Rule inv:
self.containedFormula.ocllsTypeOf(Implication) implies
(self.containedFormula.ocllsTypeOf(Implication).consequent.ocllsTypeOf(Conjunction) implies
self.containedFormula.ocllsTypeOf(Implication).consequent.
ocllsTypeOf(Conjunction).connectedFormulas→forAll(ocllsTypeOf(FMolecule) or
ocllsTypeOf(PAtom)))

5. A similar constraint is defined for rules with universal quantifier:
context Rule inv:
self.containedFormula.ocllsTypeOf(ForAll) implies
(self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
consequent.ocllsTypeOf(Conjunction) implies
self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
consequent.ocllsTypeOf(Conjunction).connectedFormulas→forAll(
ocllsTypeOf(FMolecule) or ocllsTypeOf(PAtom)))

6. The body of a rule can be any formula except a rule or a query. The following constraint defines this for rules without a universal quantifier:
context Rule inv:
self.containedFormula.ocllsTypeOf(Implication) implies
not self.containedFormula.ocllsTypeOf(Implication).antecedent.ocllsTypeOf(Rule)
and
not self.containedFormula.ocllsTypeOf(Implication).antecedent.ocllsTypeOf(Query)

7. A similar constraint is defined for rules with universal quantifier:
context Rule inv:
self.containedFormula.ocllsTypeOf(ForAll) implies
not self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
antecedent.ocllsTypeOf(Rule) and
not self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication).
antecedent.ocllsTypeOf(Query)

Three OCL constraints specify how partial formulas are combined in queries:

1. A query is an implication formula with empty head, and can have a universal quantifier in front:
context Query inv:
(self.containedFormula.ocllsTypeOf(Implication) or
self.containedFormula.ocllsTypeOf(ForAll))
and
(self.containedFormula.ocllsTypeOf(Implication) implies
self.containedFormula.ocllsTypeOf(Implication).consequent→isEmpty)

and

```
(self.containedFormula.ocllsTypeOf(ForAll) implies
self.containedFormula.ocllsTypeOf(ForAll).containedFormula.ocllsTypeOf(Implication)
and self.containedFormula.ocllsTypeOf(ForAll).containedFormula.
ocllsTypeOf(Implication).consequent→isEmpty)
```

2. The formula in the body of the query can be any formula except a rule or a query. The following constraint defines this for queries without a universal quantifier:

context Query inv:

```
self.containedFormula.ocllsTypeOf(Implication) implies
not self.containedFormula.ocllsTypeOf(Implication).consequent.ocllsTypeOf(Rule)
and not self.containedFormula.ocllsTypeOf(Implication).consequent.ocllsTypeOf(Query)
```

3. A similar constraint is defined for queries with universal quantifier:

context Query inv:

```
self.containedFormula.ocllsTypeOf(ForAll) implies
not self.containedFormula.ocllsTypeOf(ForAll).containedFormula.
ocllsTypeOf(Implication).antecedent.ocllsTypeOf(Rule)
and not self.containedFormula.ocllsTypeOf(ForAll).containedFormula.
ocllsTypeOf(Implication).antecedent.ocllsTypeOf(Query)
```

5.2.5 Logical Connectives

Partial formulas can be combined using logical connectives to form more complex formulas. Not only facts (P-atoms, F-atoms and F-molecules) but also logical quantifiers or other formulas which already connected through logical connectives, can be combined. The following table gives the F-Logic symbols and the representation in the metamodel for the logical connectives in F-Logic:

F-Logic Symbol	Metamodel element
AND	class <i>Conjunction</i>
↔	class <i>Equivalence</i>
→	class <i>Implication</i>
NOT	class <i>Negation</i>
OR	class <i>Disjunction</i>

Figure 5.8 details logical connectives⁸. Each of these subclasses is connected to the class *Formula*, specifying the formulas that are combined. *Conjunction* and *Disjunction* both have an association *connectedFormulas* with multiplicity 'two to many' in the metamodel, as they combine two or more partial formulas. *Equivalence* combines exactly two formulas, denoted by the multiplicity 'exactly two' on the association *connectedFormulas*. As a negation is only containing one formula, the association *connectedFormula* from *Negation* to *Formula* has multiplicity 'exactly one'. Finally, the metamodel provides two separate associations from *Implication* to the two formulas it combines, *antecedent* and *consequent*, as it is necessary to be able to distinguish between the two formulas as being the one before or the one behind the implication-sign. OCL constraints restrict the metamodel in the sense that the five logical connectives cannot be used in combining rules or queries:

1. A conjunction can not combine rules or queries:

context Conjunction inv:

```
self.connectedFormulas→forAll(not ocllsTypeOf(Rule) and not ocllsTypeOf(Query))
```

⁸Note that the two classes *Formula* in the figure represent the same metaclass but are only depicted twice for the sake of clarity, as allowed in UML diagrams.

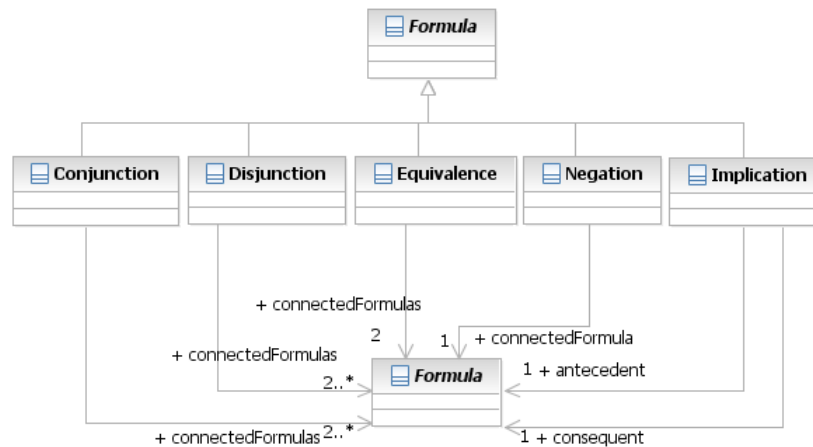


Figure 5.8: F-Logic metamodel: logical connectives

2. A disjunction can not combine rules or queries:
context Disjunction inv:
self.connectedFormulas→forall(not oclIsTypeOf(Rule) and not oclIsTypeOf(Query))
3. An equivalence construct can not combine rules or queries:
context Equivalence inv:
self.connectedFormulas→forall(not oclIsTypeOf(Rule) and not oclIsTypeOf(Query))
4. A negation can not be defined on a rule or a query:
context Negation inv:
self.connectedFormula→forall(not oclIsTypeOf(Rule) and not oclIsTypeOf(Query))
5. An implication can not combine rules or queries:
context Implication inv:
self.antecedent→forall(not oclIsTypeOf(Rule) and not oclIsTypeOf(Query)) and
self.consequent→forall(not oclIsTypeOf(Rule) and not oclIsTypeOf(Query))

5.2.6 Logical Quantifiers

F-Logic allows the two logical quantifiers \forall and \exists , binding variables in a formula. Figure 5.9 shows how the metamodel represents the two logical quantifiers as the subclasses *Exists* and *ForAll* of the metaclass *Formula*. Both are connected to *Formula* via an association *containedFormula*, specifying the formula that is combined with the quantifier. To specify the variables that are bound to the logical quantifiers, the metamodel provides an association *boundVariables* from both to the metaclass *Variable*. To make sure that a variable that is connected to a quantifier through their association *boundVariables* is also connected to the quantifier in the other direction through the association *isBoundToExists* respectively *isBoundToForAll*, and vice versa, three OCL constraints are defined:

1. When a variable is bound to a quantifier in one direction, than this quantifier must have the variable defined as one of its bound variables in the other direction:
context Variable inv:
(self.isBoundToExists=1 implies
self.isBoundToExists.boundVariables→exists(v: Variable | v=self)) and
(self.isBoundToForAll=1 implies
self.isBoundToForAll.boundVariables→exists(v: Variable | v=self))

2. When an existential quantifier has defined a variable as one of its bound variables, then this variable must be defined as bound to that quantifier:

context Exists inv:

self.boundVariables→forall(v : Variable | v.isBoundToExists=self)

3. When a universal quantifier has defined a variable as one of its bound variables, then this variable must be defined as bound to that quantifier:

context ForAll inv:

self.boundVariables→forall(v : Variable | v.isBoundToForAll=self)

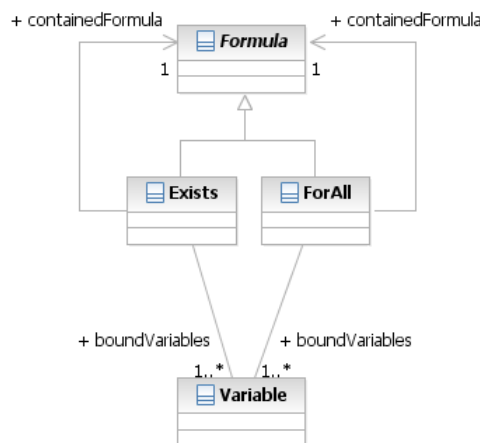


Figure 5.9: F-Logic metamodel: logical quantifiers

5.2.7 F-Atoms and F-Molecules

Let $c, c_1, c_2, m, o, r, r_1, r_2, \dots, r_n$ and t be F-Logic id-terms. Then we can define an F-atom as a fact expression with one of the following forms:

- instanceOf assertion: $o : c$ denotes that object o is an instance of class c .
- subclassOf assertion: $c_1 :: c_2$ denotes that class c_1 is a subclass of class c_2 .
- single-valued method signature (method definition): $c[m \Rightarrow t]$ is a signature-atom specifying that the application of the single-valued (or functional) method m on an object of class c has as result an object of type t , where single-valued or functional means that at most one object exists as value for the application of the method on an object.
- multi-valued method signature (method definition): $c[m \Rightarrow\Rightarrow t]$ is a multi-valued signature-atom denoting that the application of the method m on an instance of class c has a set of possible objects of type t as result.
- single-valued method application (method instantiation): $o[m \rightarrow r]$ expresses that the application of the method m on the object o has the object r as value.
- multi-valued method application (method instantiation): $o[m \rightarrow \{r_1, r_2, \dots, r_n\}]$ expresses that a set of the objects r_1, r_2, \dots, r_n is the result of the application of the method m on object o .

In the method signatures and method applications, the parts between the square brackets is referred to as the *method part*, whereas the object in front of the brackets is called the *host* in F-Logic. All method signatures and method applications can additionally have parameters.

To group several F-atoms about an object together conjunctively, F-Logic allows F-molecules. F-molecules can contain several method applications or method signatures in the method part of one construct, and moreover, not only a class but also an *instanceOf* or *subclassOf* assertion can appear as host in front of the method list⁹. For example

$$X : \text{Researcher}[\text{authorOf} \rightarrow Y; \text{cooperatesWith} \rightarrow Z]$$

is equivalent to

$$X : \text{Researcher} \wedge X[\text{authorOf} \rightarrow Y] \wedge X[\text{cooperatesWith} \rightarrow Z]$$

Ē

Figure 5.10 describes a metaclass *FMolecule* as a subclass of the metaclass *Formula* to represent an F-molecule. Because F-atoms are actually a special kind of F-molecules which can only have a simple term as host, and can only have one method in the method part, whereas F-molecules are unrestricted as regards the host objects as well as the number of methods, F-atoms are represented as a subclass of the metaclass *FMolecule*, called *FAtom*.

An association called *host* from the metaclass *FMolecule* to the metaclass *HostObject* links an F-molecule (and so an F-atom) to its host object, whereas an association called *methods* links an F-molecule (and so an F-atom) to its methods. Both F-molecules and F-atoms have always exactly one host object, which is denoted in the metamodel by the multiplicity 'exactly one' on the association *host*. The association *methods* has multiplicity 'zero to many' since an F-molecule has one or more methods, but an F-atoms has zero methods in case it has a simple term (a variable, a functional term or a constant) as host object. An OCL constraint restricts the metamodel with respect to the number of methods in F-atoms:

1. An F-atom is an F-molecule with exactly one method in case it has a simple term as host object, and zero methods otherwise:

context FAtom inv:

(self.host.ocllsTypeOf(Term) and self.methods→size(=1) or

(not self.host.ocllsTypeOf(Term) and self.methods→size(=0))

Figure 5.11 shows the representation of the host objects in the metamodel. The three different types of host objects are defined as subclasses of the abstract superclass *HostObject*. The first type of host object is a simple term, represented by the metaclass *Term* which we introduced earlier and is an abstract supertype for variables, constants and functional terms. Secondly, a host object can be a combination of two terms of which one is defined to be the subclass of the other one, represented by the metaclass *SubClassOf*. The metaclass *SubClassOf* is connected to the two terms it combines, via the associations *subclass* and *superclass*. Thirdly, the host object can be a combination of two terms of which the first one is defined to be an instance of a specific class represented by the second term. This last type of host object is represented in the metamodel by the metaclass *InstanceOf*, which is connected to the metaclass *Term* via the associations *instance* and *class* specifying the terms that are combined in the construct.

Figure 5.12 shows how the methods are represented in the metamodel. The four different types of method signatures and method applications are all represented as a subclass of the abstract class *Method*: *SingleValuedSignature*, *SingleValuedApplication*, *MultiValuedSignature* and *MultiValuedApplication*. Every type of method has a name, which is represented through the association *name* of their superclass. Additionally,

⁹Note that atoms can also be nested. For example, the value of a method application can be defined as an atom itself.

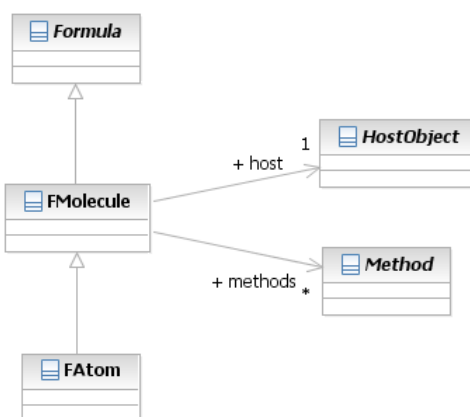


Figure 5.10: F-Logic metamodel: F-molecules

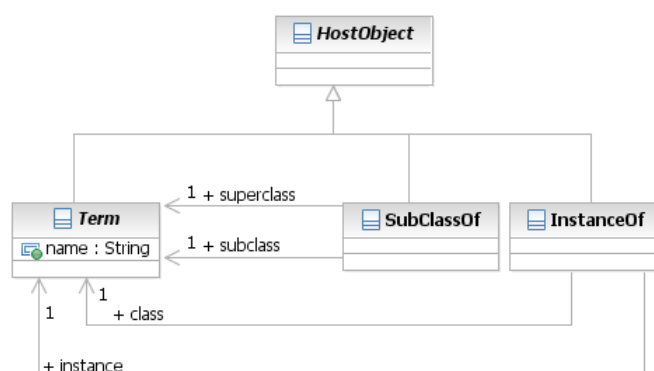


Figure 5.11: F-Logic metamodel: F-molecule host objects

the association *parameters* with multiplicity 'zero to many' represents the optional parameters a method in F-Logic can have.

The fact that all method types except the multi-valued method application have exactly one object as value, is represented by the association *value* with multiplicity 'exactly one' that the three metaclasses *SingleValuedSignature*, *SingleValuedApplication* and *MultiValuedSignature* share. As the multi-valued method application has a set of one or more method values, the association *value* between *MultiValuedApplication* and *MethodValue* has multiplicity 'one to many'¹⁰.

As not only terms but also F-molecules itself are allowed as method values, an abstract metaclass *MethodValue* is defined in the metamodel for the different types of method values. The metaclasses *Term* and *FMolecule* which were introduced already earlier, are defined as subclasses of the metaclass *MethodValue*.

5.2.8 P-Atoms

F-Logic provides P-atoms for compatibility with languages like Datalog. P-atoms consist of a predicate symbol *p* and a list of id-terms F_1 to F_n as parameters: $p(F_1, \dots, F_n)$. Information expressed by F-atoms can usually also be represented by P-atoms.

F-Logic additionally provides some built-in features, including several comparison predicates, the basic arithmetic operators, and so forth. Built-ins are actually specific, predefined P-atoms with at least one

¹⁰Note that, although a multi-valued method has more method values, the method value of a multi-valued method signature is defined as a class and hence it has not more than one method value.

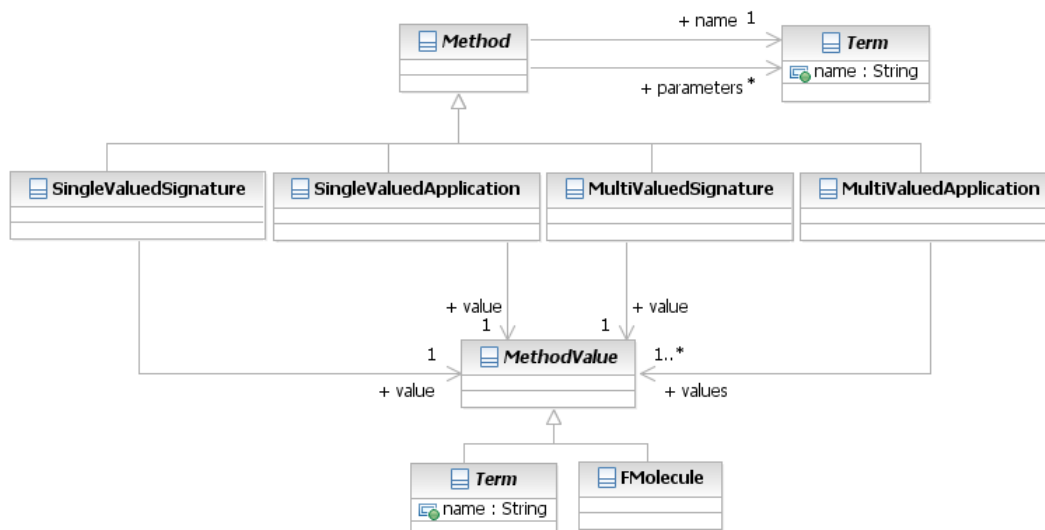


Figure 5.12: F-Logic metamodel: methods

parameter.

Figure 5.13 demonstrates the metamodel representation of P-atoms as the metaclass *PAtom*. The predicate symbol in front of the parameters in a P-atom is represented in the metamodel by the class *PredicateSymbol* which has an attribute *name* specifying the predicate's name. The metaclass *PAtom* is connected to the metaclass *PredicateSymbol* through the association *predicate* with multiplicity 'exactly one', as every P-atom must have a predicate symbol. The P-atoms's parameters are represented by instantiations of any subclass of the metaclass *Term*, to which the metaclass *PAtom* is connected with the ordered association *parameters*. This association carries a multiplicity 'zero to many' as a P-atom does not have any restrictions with respect to the number of parameters: it can contain zero but also many parameters.

As F-Logic built-ins are used in the same way as predicate symbols, the metamodel defines a metaclass *Builtin* as a subclass of the metaclass *PredicateSymbol*. An OCL constraint restricts the metamodel with respect to the number of parameters of a built-in:

1. If the predicate is a built-in, it must have at least one parameter:
 context PAtom inv:
 self.predicate.oclsTypeOf(Builtin) implies self.parameters→size()>0

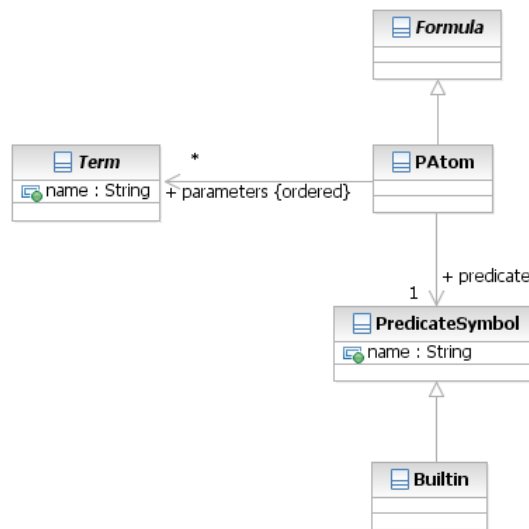


Figure 5.13: F-Logic metamodel: P-atoms

Chapter 6

The Mapping Metamodel

When people are modeling the same domain, they mostly produce different results, even when then use the same language. Mappings have to be defined between these ontologies to achieve an interoperation between applications or data relying on these ontologies. This chapter provides an extension for the metamodel for OWL and SWRL to give additional support for mappings between heterogeneous ontologies.

Section 6.1 introduces the metamodel extension in the same way as we did in the introduction of the metamodel for OWL and SWRL, and for F-Logic. While introducing the various mapping aspects¹, we discuss their representation in the metamodel. Accompanying UML diagrams document the understanding of the metamodel.²

The metamodel is a common metamodel for the different OWL mapping languages. On top of this common metamodel, we define two sets of constraints to concretize it to two specific OWL ontology mapping languages, DL-safe mappings and C-OWL. Section 6.2 starting on page 66 presents the extension for C-OWL mappings, consisting of a set of constraints. Similarly, Section 6.3 starting on page 67 presents the extension for DL-Safe Mappings.

6.1 A Common MOF-based Metamodel Extension for OWL Ontology Mappings

This section presents the common metamodel extension for OWL ontology mappings in two subsections: Section 6.1.1 presents mappings, after which Section 6.1.2 presents queries.

6.1.1 Mappings

We use a mapping architecture that has the greatest level of generality in the sense that other architectures can be simulated. In particular, we make the following choices:

- A mapping is a set of mapping assertions that consist of a semantic relation between mappable elements in different ontologies. Figure 6.1 demonstrates how this structure is represented in the metamodel by the five meta-classes *Mapping*, *MappingAssertion*, *Ontology*, *SemanticRelation* and *MappableElement* and their associations.
- Mappings are first-class objects that exist independent of the ontologies. Mappings are directed, and there can be more than one mapping between two ontologies. The direction of a mapping is defined

¹Remember, however, that the OWL ontology mapping languages and their general aspects, are not part of our contribution.

²In doing so, meta-classes that are colored or carry a little icon again denote elements from the metamodel for OWL or SWRL.

through the associations *sourceOntology* and *targetOntology* of the metaclass *Mapping*, as the mapping is defined from the source to the target ontology. The cardinalities on both associations denote that to each *Mapping* instantiation, there is exactly one *Ontology* connected as source and one as target.

These choices leave us with a lot of freedom for defining and using mappings. For each pair of ontologies, several mappings can be defined or, in case of approaches that see mappings as parts of an ontology, only one single mapping can be defined. Bi-directional mappings can be described in terms of two directed mappings.

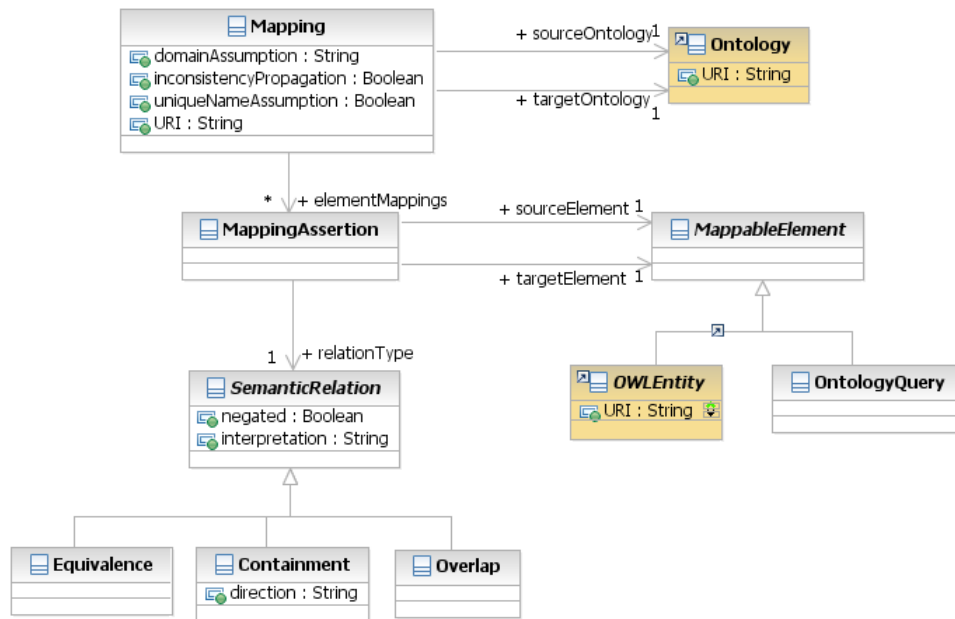


Figure 6.1: OWL mapping metamodel: mappings

The central class in the mapping metamodel, the class *Mapping*, is given four attributes. For the assumptions about the domain, the metamodel defines an attribute *DomainAssumption*. This attribute may take specific values that describe the relationship between the connected domains: *overlap*, *containment* (in either direction) or *equivalence*.

The question of what is preserved by a mapping is tightly connected to the hidden assumptions made by different mapping formalisms. A number of important assumptions that influence this aspect have been identified and formalized in [SSW05a]. A first basic distinction concerns the relationship between the sets of objects (domains) described by the mapped ontologies. Generally, we can distinguish between a global domain and local domain assumption:

Global Domain assumes that both ontologies describe exactly the same set of objects. As a result, semantic relations are interpreted in the same way as axioms in the ontologies. This domain assumption is referred to as *equivalence*, whereas there are special cases of this assumption, where one ontology is regarded as a global schema and describes the set of all objects, other ontologies are assumed to describe subsets of these objects. Such domain assumption is called *containment*.

Local Domains do not assume that ontologies describe the same set of objects. This means that mappings and ontology axioms normally have different semantics. There are variations of this assumption in the sense that sometimes it is assumed that the sets of objects are completely disjoint and sometimes they are assumed to overlap each other, represented by the domain assumption called *Overlap*.

These assumptions about the relationship between the domains are especially important for extensional mapping definitions, because in cases where two ontologies do not talk about the same set of instances, the extensional interpretation of a mapping is problematic as classes that are meant to represent the same aspect of the world can have disjoint extensions.

The second attribute of the metaclass *Mapping* is called *inconsistencyPropagation*, and specifies whether the mapping propagates inconsistencies across mapped ontologies. *uniqueNameAssumption*, the third attribute of the metaclass *Mapping*, specifies whether the mappings are assumed to use unique names for objects, an assumption which is often made in the area of database integration. The fourth attribute, *URI*, is an optional URI which allows to uniquely identify a mapping and refer to it as a first-class object.

The set of mapping assertions of a mapping is denoted by the relationship between the two classes *Mapping* and *MappingAssertion*. The elements that are mapped in a *MappingAssertion* are defined by the class *MappableElement*. A *MappingAssertion* is defined through exactly one *SemanticRelation*, one source *MappableElement* and one target *MappableElement*. This is defined through the three associations starting from *MappingAssertion* and their cardinalities.

A number of different kinds of semantic relations have been proposed for mapping assertions and are represented as subclasses of the abstract superclass *SemanticRelation*:

Equivalence (\equiv) Equivalence, represented by the metaclass *Equivalence*, states that the connected elements represent the same aspect of the real world according to some equivalence criteria. A very strong form of equivalence is equality, if the connected elements represent exactly the same real world object. Specific forms of the equivalence relation are to be defined as subclasses of *Equivalence* in the specific metamodels of the concrete mapping formalisms.

Containment (\sqsubseteq, \sqsupseteq) Containment, represented by the metaclass *Containment*, states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. Depending on which of the elements is more specific, the containment relation is defined in the one or in the other direction. This direction is specified in the metamodel by the attribute *direction*, which can be *sound* (\sqsubseteq) or *complete* (\sqsupseteq). If this attribute value is *sound*, the source element is more specific element than the target element. In case of the attribute value *complete*, it is the other way around, thus the target element is more specific than the source element.

Overlap (\circ) Overlap, represented by the metaclass *Overlap*, states that the connected elements represent different aspects of the world, but have an overlap in some respect. In particular, it states that some objects described by the element in the one ontology may also be described by the connected element in the other ontology.

In some approaches, these basic relations are supplemented by their negative counterparts, for which the metamodel provides an attribute *negated* for the abstract superclass *SemanticRelation*. For example, a negated *Overlap* relation specifies the disjointness of two elements. The corresponding relations can be used to describe that two elements are *not* equivalent (\neq), *not* contained in each other ($\not\sqsubseteq$) or *not* overlapping or disjoint respectively (\emptyset). Adding these negative versions of the relations leaves us with eight semantic relations that cover all existing proposals for mapping languages.

In addition to the type of semantic relation, an important distinction is whether the mappings are to be interpreted as extensional or as intensional relationships, specified through the attribute *interpretation* of the metaclass *SemanticRelation*.

Extensional The extension of a concept consists of the things which fall under the concept. In extensional mapping definitions, the semantic relations are interpreted as set-relations between the sets of objects represented by elements in the ontologies. Intuitively, elements that are extensionally the same have to represent the same set of objects.

Intensional The intension of a concept consists of the qualities or properties which go to make up the concept. In the case of intensional mappings, the semantic relations relate the concepts directly, i.e. considering the properties of the concept itself. In particular, if two concepts are intensionally the same, they refer to exactly the same real world concept.

As mappable elements, the metamodel contains the class *OWLEntity* that represents an arbitrary part of an ontology specification. While this already covers many of the existing mapping approaches, there are a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between (conjunctive) queries to connect models, which leads to a considerably increased expressiveness. These queries are represented by the metaclass *OntologyQuery*. Note that the metamodel in principle supports all semantic relations for all mappable elements. OCL constraints for specific mapping formalisms can restrict the combinations of semantic relations and mappable elements.

6.1.2 Queries

A mapping assertion can take a query as mappable element. Figure 6.2 demonstrates the class *Query* that reuses constructs from the SWRL metamodel.

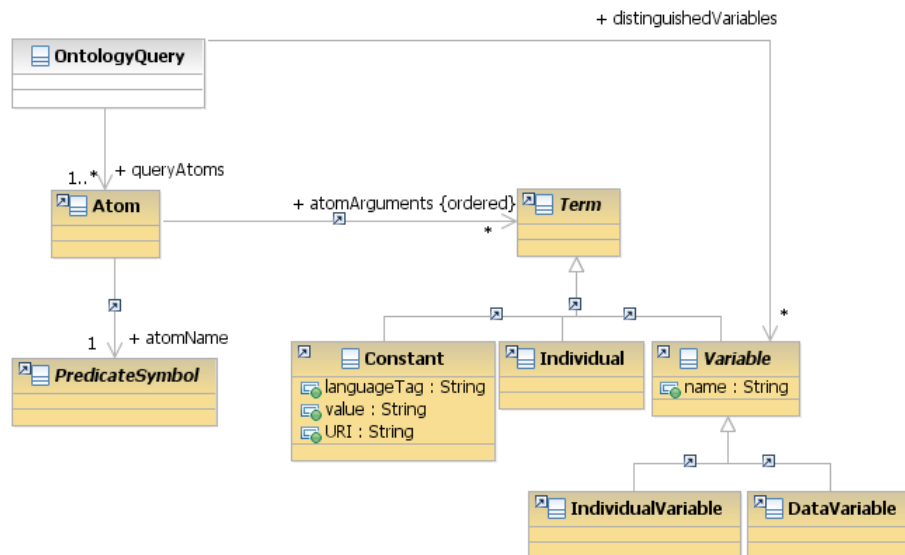


Figure 6.2: OWL mapping metamodel: queries

We reuse large parts of the rule metamodel as conceptual rules and queries are of very similar nature [TF05]: A rule consists of a rule body (antecedent) and rule head (consequent), both of which are conjunctions of logical atoms. A query can be considered as a special kind of rule with an empty head. The distinguished variables specify the variables that are returned by the query. Informally, the answer to a query consists of all variable bindings for which the grounded rule body is logically implied by the ontology. A *Query* atom also contains a *PredicateSymbol* and some, possibly just one, *Terms*. In the SWRL metamodel, we defined the permitted predicate symbols through the subclasses *Description*, *DataRange*, *DataProperty*, *ObjectProperty* and *BuiltIn*. Similarly, the different types of terms, *Individual*, *Constant*, *IndividualVariable* and *DataVariable* are specified as subclasses of *Term*. Distinguished variables of a query are differentiated through an association between *Query* and *Variable*. An OCL constraint a restriction on the use of distinguished variables:

1. A variable can only be a distinguished variable of a query if it is a term of one of the atoms of the query:
`self.distinguishedVariables→forall(v: Variable | self.queryAtoms→exists(a: Atom | a.atomArguments→exists(v | true)))`

6.2 OCL Constraints for C-OWL

We define OCL constraints on the common mapping metamodel extension to concretize it according to the specific formalism C-OWL [BGvH⁺03]. We list the specific characteristics of C-OWL and introduce the necessary constraints for them. For each constraint, firstly the context of the constraint, so the class in the metamodel on which it is to be defined, is defined using the OCL syntax "context *classname* inv."³. Some existing reasoners support only a subset of C-OWL. Additional constraints could be defined to support this.

1. C-OWL does not have unique name assumption. To reflect this in the metamodel, a constraint defines that the value of the attribute *uniqueNameAssumption* of the class *Mapping* is always 'false':
context Mapping inv:
self.uniqueNameAssumption = false
2. C-OWL does not have inconsistency propagation. Similarly, a constraint is defined to set the value of the attribute *inconsistencyPropagation* of the class *Mapping* to 'false':
context Mapping inv:
self.inconsistencyPropagation = false
3. The relationship between the connected domains of a mapping in C-OWL is always assumed to be *overlap*. A constraint sets the value of the attribute *domainAssumption* of the class *Mapping* to 'overlap':
context Mapping inv:
self.domainAssumption = 'overlap'
4. C-OWL does not allow to define mappings between queries. Moreover, only object properties, classes and individuals are allowed as mappable elements. A constraint defines that any mappable element in a mapping must be an *ObjectProperty*, an *OWLClass* or an *Individual*:
context MappableElement inv:
self.oclsTypeOf(ObjectProperty) or
self.oclsTypeOf(OWLClass) or
self.oclsTypeOf(Individual)
5. A mapping assertion can only be defined between elements of the same kind. As the previous constraint defines that mappings can only be defined between three specific types of elements, an additional constraint can easily define that when the source element is one of these specific types, then the target elements must be of that type as well:
context MappingAssertion inv:
(self.sourceElement.oclsTypeOf(OWLClass) implies
self.targetElement.oclsTypeOf(OWLClass)) and
(self.sourceElement.oclsTypeOf(ObjectProperty) implies
self.targetElement.oclsTypeOf(ObjectProperty)) and
(self.sourceElement.oclsTypeOf(Individual) implies
self.targetElement.oclsTypeOf(Individual))
6. As semantic relations in mappings, C-OWL supports *equivalence*, *containment* (sound as well as complete), *overlap* and *negated overlap* (called *disjoint*). A constraint defines this by specifying which different subclasses of *SemanticRelation* are allowed. When only the non-negated version of the semantic relation is allowed, the constraint defines that the attribute *negated* of the class *SemanticRelation* is set to 'false':
context SemanticRelation inv:
(self.oclsTypeOf(Equivalence) and self.negated = false) or

³where 'inv' stands for the constraint type *invariant*.

(self.oclsTypeOf(Containment) and self.negated = false) or
self.oclsTypeOf(Overlap)

6.3 OCL Constraints for DL-Safe Mappings

This section provides OCL constraints on the common metamodel for OWL ontology mappings to concretize it to the formalism DL-Safe Mappings [HM05]. Again, we highlight the specific characteristics of the language and provide the appropriate constraints.

1. DL-Safe Mappings do not have unique name assumption. To reflect this in the metamodel, a constraint defines that the value of the attribute *uniqueNameAssumption* of the class *Mapping* is always 'false':
context Mapping inv:
self.uniqueNameAssumption = false
2. DL-Safe Mappings always have inconsistency propagation. A constraint is defined to set the value of the attribute *inconsistencyPropagation* of the class *Mapping* to 'true':
context Mapping inv:
self.inconsistencyPropagation = true
3. The relationship between the connected domains of a DL-Safe Mapping is always assumed to be *equivalence*. A constraint sets the value of the attribute *domainAssumption* of the class *Mapping* to 'equivalence':
context Mapping inv:
self.domainAssumption = 'equivalence'
4. DL-Safe Mappings support mappings between queries, properties, classes, individuals and datatypes. A constraint defines that the type of a *MappableElement* must be one of these subclasses:
context MappableElement inv:
self.oclsTypeOf(OntologyQuery) or
self.oclsTypeOf(ObjectProperty) or
self.oclsTypeOf(DataProperty) or
self.oclsTypeOf(OWLClass) or
self.oclsTypeOf(Individual) or
self.oclsTypeOf(Datatype)
5. In DL-Safe Mappings, elements being mapped to each other must be of the same kind. Thus, when one wants to map for instance a concept to a query, the concept should be modelled as a query. A constraint defines that when the source element is of a specific type, the target element must be of the same type:
context MappingAssertion inv:
(self.sourceElement.oclsTypeOf(OntologyQuery) implies
self.targetElement.oclsTypeOf(OntologyQuery)) and
(self.sourceElement.oclsTypeOf(ObjectProperty) implies
self.targetElement.oclsTypeOf(ObjectProperty)) and
(self.sourceElement.oclsTypeOf(DataProperty) implies
self.targetElement.oclsTypeOf(DataProperty)) and
(self.sourceElement.oclsTypeOf(OWLClass) implies
self.targetElement.oclsTypeOf(OWLClass)) and
(self.sourceElement.oclsTypeOf(Individual) implies
self.targetElement.oclsTypeOf(Individual)) and
(self.sourceElement.oclsTypeOf(Datatype) implies
self.targetElement.oclsTypeOf(Datatype))

6. DL-Safe Mappings specify that queries that are mapped to each other, must contain the same distinguished variables. A constraint defines that when both elements are a query, each variable that exists as distinguished variable in the source element, must exist as distinguished variable in the target element:
context MappingAssertion inv:
self.sourceElement.ocllsTypeOf(Query) and
self.targetElement.ocllsTypeOf(Query) implies
self.sourceElement.ocllAsType(Query).distinguishedVariables→
forAll(v: Variable | self.targetElement.ocllAsType(Query).distinguishedVariables→
exists(v | true))
7. The interpretation of semantic relations in DL-Safe Mappings is always assumed to be extensional. A constraint defines that the value of the attribute *interpretation* of the class *SemanticRelation* must be set to 'extensional':
context SemanticRelation inv:
self.interpretation = 'extensional'
8. DL-Safe Mappings support the semantic relations *equivalence* and *containment* (sound as well as complete). A constraint specifies this by defining which types *SemanticRelation* can have and what its value for the attribute *negated* should be:
context SemanticRelation inv:
(self.ocllsTypeOf(Equivalence) and self.negated = false) or
(self.ocllsTypeOf(Containment) and self.negated = false)

Chapter 7

The Metamodel for Modular Ontologies

In software engineering, a modular software is a program in which significant parts, modules, are identified. A module generally plays a particular role in the whole program that, when combined with the other modules, contributes to the objective of the overall software. Well known advantages of modular design include clarity, reusability, and extensibility: a module is designed to be an intrinsically self-contained and reusable component, defined and managed independently from the programs it is intended to be included in.

It is a common opinion that ontology engineering shares a lot with software engineering, and the advantages of having modular ontologies instead of big and monolithic ones are easy to understand. Ontology modules are made to be reusable knowledge components, facilitating collaborative design and maintenance within a network of ontologies. Therefore, the goal of this chapter is to come up with a language for defining and managing ontology modules, as a part of the NeOn metamodel. We here directly build on the OWL metamodel for the specification of the content of a module and the mapping metamodel to relate the content of heterogeneous modules.

7.1 Design Considerations

Modularization is essential to support collaborative ontology editing and browsing, reuse and selection. Within the networked scenario, ontologies are mainly built in distributed locations, which means naturally, they are modules in a network of ontologies. At the same time, ontologies are difficult to manage especially when they are interconnected in a network. Interdependencies between modules of interconnected ontologies have to be made clear so that these modules can be easily managed (e.g., reused, shared). Ontology selection and reuse will play a major part in the future of the Semantic Web because with the increasing number of ontologies available over the internet, people will more likely use ontologies just like program libraries by selecting and reusing ontology modules [dSM06]. Modularization will offer the possibility for a more fine-grained sharing mechanism where ontology modules are well encapsulated and ready for future reuse and development. Many formalisms for handling ontology modules have been proposed [BS03, SK03, SP04], however, they have their own limitations in some scenarios [SR06, BCH06b].

7.1.1 Existing Formalisms for Ontology Modularization

In the following we provide an overview of several modular ontology formalisms.

Modularization with OWL Import The OWL ontology language provides limited support to modular ontologies: an ontology document – identified via its ontology URI – can be imported by another document using the `owl:imports` statement. The semantics of this import statement is that all definitions contained in the imported ontology document are included in the importing ontology, as if they were defined in the importing ontology document. It is worth to mention that `owl:imports` is directed –only the importing ontology is

affected– and transitive –if A imports B and B imports C , then A also imports the definitions contained in C . Moreover, cyclic imports are allowed (e.g. A imports B and B imports A).

One of the most commonly mentioned weaknesses of the importing mechanism in OWL is that it does not provide any support for partial import [VOM02, PSZ06]. Even if only a part of the imported ontology is relevant or agreed in the importing ontology, every definition is included. Moreover, there is no logical difference between imported definitions and proper definitions in the importing ontology: they share the same interpretations.

Distributed Description Logics Unlike import mechanisms that include elements from some modules into the considered one, Distributed Description Logics (DDLs) [BS02] adopt a *linking mechanism*, relating the elements of "local ontologies" (called *context*) with elements of external ontologies (contexts). Each context M_i is associated to its own local interpretation. Semantic relations are used to draw correspondences between elements of local interpretation domains. These relations are expressed using *bridge rules* of the form:

- $i : \phi \xrightarrow{\text{E}} j : \psi$ (*into rule*), with semantics: $r_{ij}(\phi^{I_i}) \subseteq \psi^{I_j}$
- $i : \phi \xrightarrow{\text{O}} j : \psi$ (*onto rule*), with semantics: $r_{ij}(\phi^{I_i}) \supseteq \psi^{I_j}$

where $I_i = (\Delta_i, \mathcal{I}_i)$ (respectively $I_j = (\Delta_j, \mathcal{I}_j)$) is the local interpretation of M_i (respectively M_j), ϕ and ψ are formulae, and r_{ij} is a *domain relation* mapping elements of the interpretation domain of M_i to elements of the interpretation domain of M_j ($r_{ij} \subseteq \Delta_i \times \Delta_j$). We only discuss bridge rules between concepts (meaning that ϕ and ψ are concept names or expressions) since it is the only case that has reported reasoning support [ST05].

Bridge rules between concepts cover one of the most important scenarios in modular ontologies: they are intended to assert relationships, like concept inclusions, between elements of two different local ontologies. However, mainly because of the local interpretation, they are not supposed to be read as classical DL axioms. In particular, a bridge rule only affects the interpretation of the target element, meaning for example that $i : \phi \xrightarrow{\text{E}} j : \psi$ is not equivalent to $j : \psi \xrightarrow{\text{O}} i : \phi$.

Arbitrary domain relations may not preserve concept unsatisfiability among different contexts which may result in some reasoning difficulties [BCH06c]. Furthermore, while subset relations (between concept interpretations) is transitive, DDL domain relations are not transitive, therefore bridge rules cannot be *transitively reused* by multiple contexts. Those problems are recently recognized in several papers [BCH06b, BCH06c, SSW05b, SSW05a] and it is proposed that arbitrary domain relations should be avoided. For example, domain relations should be one-to-one [SSW05a, BCH06c] and non-empty [SSW05b].

Integrity and Change of Modular Terminologies in DDL Influenced by DDL semantics, [SK03] adopts a view-based information integration approach to express relationships between ontology modules. In particular, in this approach ontology modules are connected by conjunctive queries. This way of connecting modules is more expressive than simple one-to-one mappings between concept names but less expressive than the logical language used to describe concepts.

[SK03] defines an ontology module – abstracted from a particular ontology language – as a triple $M = (C, \mathcal{R}, \mathcal{O})$, where C is a set of concept definitions, \mathcal{R} is a set of relation definitions and \mathcal{O} is a set of object definitions. A conjunctive query Q over an ontology module $M = (C, \mathcal{R}, \mathcal{O})$ is defined as an expression of the form $q_1 \wedge \dots \wedge q_m$, where q_i is a query term of the form $C(x)$, $R(x, y)$ or $x = y$, C and R concept and role names, respectively, and x and y are either variable or object names.

In a modular terminology it is possible to use conjunctive queries to define concepts in one module in terms of a query over another module. For this purpose, the set of concept definitions C is divided into two disjoint sets of internally and externally defined concepts C_I and C_E , respectively, with $C = C_I \cup C_E$, $C_I \cap C_E = \emptyset$.

An *internal concept* definition is specified using regular description logics based concept expressions with the form of $C \sqsubseteq D$ or $C \equiv D$, where C and D are atomic and complex concepts, respectively.

An *external concept* definition is an axiom of the form $C \equiv M : Q$, where M is a module and Q is a conjunctive query over M . It is assumed that such queries can be later reduced to complex concept descriptions using the query-rollup techniques from [HT00] in order to be able to rely on standard reasoning techniques. A modular ontology is then simply defined as a set of modules that are connected by external concept definitions. The semantics of these modules is defined using the notion of a distributed interpretation as introduced in the previous paragraph.

Although the definition of a module, in its abstract form shown above, may allow arbitrary concept, relation and object definitions, only concept definitions is studied in [SK03]. This is due to the focus of the approach to improve terminological reasoning with modular ontologies by precompiling implied subsumption relations. In that sense it can be seen as a restricted form of DDLs that enables improved efficiency for special T-Box reasoning tasks.

\mathcal{E} -Connection While DDLs are focused on one type of relation between module elements, concept inclusion, the \mathcal{E} -connection approach [KLWZ03, GPS04] allows to define *link properties* from one module to another. For example, if a module M_1 contains a concept named 1 : *Fish* and a module M_2 contains a concept named 2 : *Region*, one can connect these two modules by defining a link property named *livesIn* between 1 : *Fish* and 2 : *Region*.

Formally, given ontology modules $\{L_i\}$, a (one-way binary) link $E \in \mathcal{E}_{ij}$, where $\mathcal{E}_{ij}, i \neq j$ is the set of all links from the module i to the module j , can be used to construct a concept in module i , with the syntax and semantics specified as follows:

- $\langle E \rangle(j : C)$ or $\exists E.(j : C) : \{x \in \Delta_i | \exists y \in \Delta_j, (x, y) \in E^M, y \in C^M\}$
- $\forall E.(j : C) : \{x \in \Delta_i | \forall y \in \Delta_j, (x, y) \in E^M \rightarrow y \in C^M\}$
- $\leq nE.(j : C) : \{x \in \Delta_i | \#\{y \in \Delta_j | (x, y) \in E^M, y \in C^M\} \leq n\}$
- $\geq nE.(j : C) : \{x \in \Delta_i | \#\{y \in \Delta_j | (x, y) \in E^M, y \in C^M\} \geq n\}$

where $M = \langle \{m_i\}, \{E^M\}_{E \in \mathcal{E}_{ij}} \rangle$ is a model of the \mathcal{E} -connected ontology, m_i is the local model of L_i ; C is a concept in L_j , with interpretation $C^M = C^{m_j}$; $E^M \subseteq \Delta_i \times \Delta_j$ is the interpretation of a \mathcal{E} -connection E .

\mathcal{E} -connection restricts the terms of modules, as well as their local domains, to be disjoint. This can be a serious limitation in some scenarios, particularly because, to enforce domain disjointness, subclass relations cannot be declared between concept of two different modules.

P-DL **P-DL**, Package-based Description Logics [BCH06d], uses importing relations to connect local models. In contrast to OWL, which forces the model of an imported ontology be completely embedded in a global model, the P-DL importing relation is *partial* in that only commonly shared terms are interpreted in the overlapping part of local models. The semantics of P-DL is given as the follows: the *image domain relation* between local interpretations \mathcal{I}_i and \mathcal{I}_j (of package P_i and P_j) is $r_{ij} \subseteq \Delta_i \times \Delta_j$. P-DL importing relation is:

- one-to-one: for any $x \in \Delta_i$, there is at most one $y \in \Delta_j$, such that $(x, y) \in r_{ij}$, and vice versa.
- compositionally consistent: $r_{ij} = r_{ik} \circ r_{jk}$, where \circ denotes function composition. Therefore, semantic relations between terms in i and terms in k can be inferred even if k doesn't directly import terms from i .

Thus, a P-DL model is a virtual model constructed from partially overlapping local models by merging "shared" individuals.

P-DL also supports selective knowledge sharing by associating ontology terms and axioms with “scope limitation modifiers (SLM)”. A SLM controls the visibility of the corresponding term or axiom to entities on the web, in particular, to other packages. The scope limitation modifier of a term or an axiom t_K in package K is a boolean function $f(p, t_K)$, where p is a URI of an entity, the entity identified by p can access t_K iff $f(p, t) = true$. For example, some representative SLMs can be defined as follows:

- $\forall p, public(p, t) := true$, means t is accessible everywhere.
- $\forall p, private(p, t) := (t \in p)$, means t is visible only to its home package.

P-DL semantics ensures that distributed reasoning with a modular ontology will yield the same conclusion as that obtained by a classical reasoning process applied to an integration of the respective ontology modules [BCH06c]. However, the reported result [BCH06a] only supports reasoning in P-DL as extensions of the \mathcal{ALC} DL. Reasoning algorithms for more expressive P-DL TBox, as well as for ABox reasoning, are still to be investigated.

7.1.2 Requirements for a Module Definition Languages

A Module is an Ontology. As shown in the above overview, there is generally no clear distinction between the notion of ontology and the one of ontology module. A modular ontology is made of smaller local ontologies that can be seen as self-contained and inter-related modules, combined together for covering a broader domain. Indeed, an ontology is not inherently a module, but rather plays the role of a module for other ontologies because of the way it is related to them in an ontology network. In other terms, an ontology module is a self-contained ontology, seen according to a particular perspective, namely *reusability*. The content of an ontology module does not differ from the one of an ontology, but a module should come with additional information about how to reuse it, and how it reuse other modules.

Encapsulation / Information Hiding. The idea of encapsulation is crucial in modular software development, but has not really been studied and implemented in the domain of ontologies yet. In software engineering, it relies on the distinction between the implementation, i.e. internal elements manipulated by the developer of the module, and the interface, i.e. the elements that are exposed to be reused. This distinction between interface and implementation cannot be clearly stated when using ontology technologies like OWL. However, the essential role of a module interface is to guide the reusability of the module, by exposing reusable elements and hiding intermediary internal ones (the “implementation details”). By defining the set of reusable entities of an ontology module (the *export interface*), the developer of this module provides entry points to it, and clearly states which are the elements that can be “safely reused”. Elements that are hidden behind the interface can then evolve, be re-designed or changed, without affecting the importing modules relying on this export interface.

Partial Import. As already mentioned, the `owl:imports` mechanism has been criticized in several papers for being “global” (see e.g. [VOM02, PSZ06]): it is not possible when using this mechanism to import only the relevant and useful elements in the importing ontology. Allowing partial import has many advantages, among which *scalability* is probably the most obvious. For this reason, some intermediary solutions have been recently proposed, using, prior to import, ontology partitioning [SK04, GPSK05] techniques or some forms of reduction to a sub-vocabulary [SR06, Stu06, dSM06]. We believe that the set of elements that are used in an importing module should be explicitly stated in the module definition, so that the influence of the imported module is clarified. The semantics of the module definition language should reflect the idea of partial import by “ignoring” the definitions that are not related to the imported elements (the *import interface*), preventing the importing module to deal with irrelevant knowledge, and giving the developer of such a module the possibility to disagree with some part of the imported modules.

Links Between Modules. The formalisms for modular ontologies presented in the previous section can be divided in two main approaches: importing and linking. The previous requirements are focused on the importing approach, whereas languages like C-OWL and \mathcal{E} -connection exclusively deal with the linking approach. In the NeOn framework, these two aspects are relevant, and should be considered together: even when they are not imported, elements from different modules can be related through mappings. The NeOn metamodel already provide the required elements for expressing mappings, and these can easily be considered as a part of the content of an ontology module. However, it is important to take this aspect into account when designing the semantics of the module description language, as well as the operations for manipulating modules, so that the two approaches, importing and linking, are well integrated. Indeed, scenarios where, for example, there exist mappings between imported modules are not hard to imagine.

In this section we briefly recapitulate the basics of NeOn model for modular ontologies as defined in Deliverable D1.1.3 [dHR⁺08].

7.1.3 Abstract Definition and Notation

We start by defining sets of identifiers being used for unambiguously referring to ontology modules and mappings that might be distributed over the Web. Obviously, in practice, IRIs will be used for this purpose. So we let

- Id_{Modules} be a set of MODULE IDENTIFIERS and
- Id_{Mappings} be a set of MAPPING IDENTIFIERS, where a mapping is a set of relations (correspondences) between entities of two different ontologies.

Next we introduce generic sets describing the used ontology language. They will be instantiated depending on the concrete ontology language formalism used (e.g., OWL). Hence, let:

- Nam be a set of NAMED ELEMENTS.
In the case of OWL, Nam will be thought to contain all class names, property names and individual names.
- $Elem$ be a the set of ONTOLOGY ELEMENTS.
In the OWL case $Elem$ would contain e.g. all complex class descriptions. Clearly, $Elem$ will depend on Nam (or roughly speaking: Nam delivers the “building blocks” for $Elem$).
- We use $L : 2^{Nam} \rightarrow 2^{Elem}$ to denote the function assigning to each set P of named elements the set of ontology elements which can be generated out of P by the language constructs¹,
- For a given set O of ontology axioms, let $\text{Sig}(O)$ denote the set of named elements occurring in O , so it represents those elements the axioms from O deal with.

Having stipulated those basic sets in order to describe the general setting, we are now ready to state the notion of an ontology module on this abstract level.

Definition 5 An ONTOLOGY MODULE \mathcal{OM} is a tuple $\langle id, Imp, \mathcal{I}, M, O, E \rangle$ where

- $id \in Id_{\text{Modules}}$ is the identifier of \mathcal{OM}
- $Imp \subseteq Id_{\text{Modules}}$ is a set of identifiers of imported ontology modules (referencing those other modules whose content has to be (partially) incorporated into the module),
- \mathcal{I} is set $\{I_{id}\}_{id \in Imp}$ of IMPORT INTERFACES, with $I_{id} \subseteq Nam$ (characterizing which named elements from the imported ontology modules will be “visible” inside \mathcal{OM}),

¹In most cases – and in particular for OWL – $L(P)$ will be infinite, even if P is finite.

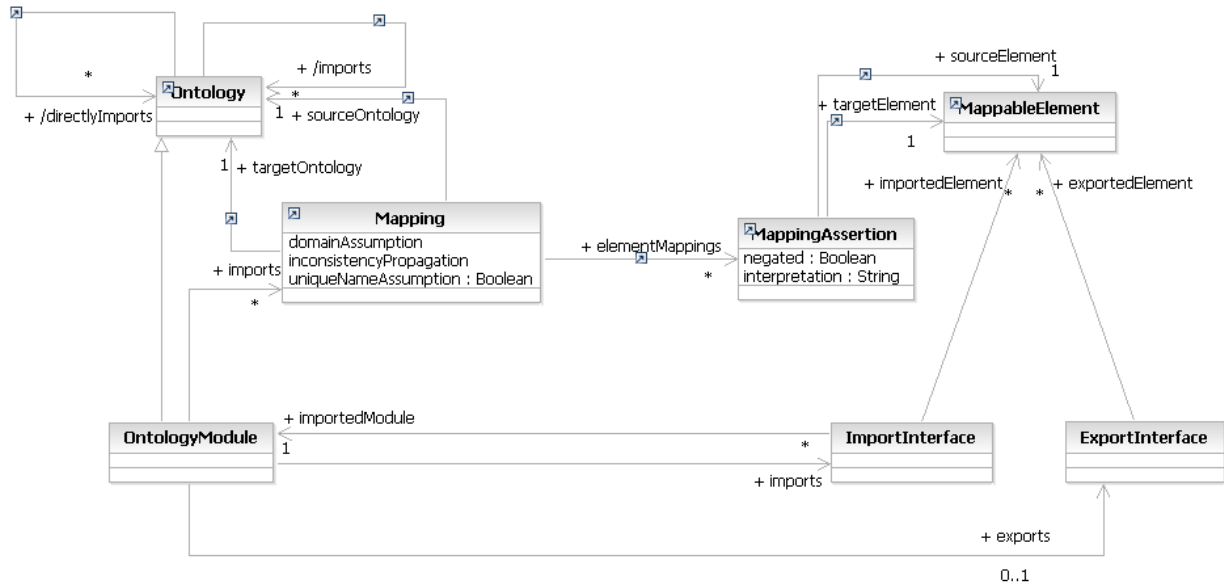


Figure 7.1: Metamodel extensions for ontology modules.

- $M \subseteq Id_{\text{Mappings}}$ is a set of identifiers of imported mappings (referencing – via mapping identifiers – those mappings between ontology modules, which are to be taken into account in \mathcal{OM}),
- O is a set of ONTOLOGY AXIOMS (hereby constituting the actual content of the ontology),
- $E \subseteq \text{Sig}(O) \cup \bigcup_{id \in Imp} \mathcal{I}_{id}$ is called EXPORT INTERFACE (telling which named entities from the ontology module are “published”, i.e., can be imported by other ontology modules).

Note that, in order to simplify the notation, we will not specify explicitly an identifier for the module: a module \mathcal{OM}_i will be considered as implicitly having “ \mathcal{OM}_i ” as identifier and will so be written: $\mathcal{OM}_i = \langle Imp_i, \mathcal{I}_i, M_i, O_i, E_i \rangle$.

In a further step we formally define the term mapping (which is supposed to be a set of directed links, correspondences, between two ontology modules establishing semantic relations between their entities).

Definition 6 A MAPPING M is a tuple $\langle s, t, C \rangle$ with

- $s, t \in Id_{\text{Modules}}$, with s being the identifier of the source ontology module and t being the identifier of the target ontology module,
- C is a set of CORRESPONDENCES of the form $e_1 \rightsquigarrow e_2$ with $e_1, e_2 \in Elem$ and $\rightsquigarrow \in R$ for a fixed set R of CORRESPONDENCE TYPES²

7.1.4 Metamodel

We propose a generic metamodel for modular ontologies according to the design considerations discussed above. The metamodel is a consistent extension of the metamodels for OWL 2 ontologies and mappings presented in the previous chapters.

²In accordance with the NeOn metamodel, this set will be fixed to $R = \{\sqsubseteq, \supseteq, \equiv, \perp, \sqsubset, \sqsupset, \neq, \Delta\}$

Figure 7.1 shows elements of the metamodel for modular ontologies. The central class in the metamodel is the class `OntologyModule`. A module is modeled as a specialization of the class `Ontology`. The intuition behind this modeling decision is that every module is also considered an ontology, enriched with additional features. In other words, a module can also be seen as a role that a particular ontology plays. In addition, an ontology provides (at most) one `ExportInterface` and a set of `ImportInterface`. The interfaces define the elements that are exposed by the imported module and reused by the importing module. The elements that can be reused by modules are `MappableElements` (defined in the mapping metamodel). A mappable element is either an `Entity` or a `Query` over an ontology, meaning that entities can be exposed in interfaces either directly or as the results of queries.

The export interface, modeled via the `exports` association, exposes the set of `MappableElements` that are intended to be reused by other modules.

The reuse of elements from one module by another module is represented via the `ImportInterface`. The association `imports` relates the importing module with the definition of the imports interface. The association `importedModule` refers to the module that is being imported, while the association `importedElement` refers to the element in the imported ontology being reused. In this sense, the `ImportInterface` can be seen to realize the ternary relationship between the importing ontology, the imported ontology, and the elements to be reused.

Additionally, an `OntologyModule` also provides an `imports` relationship with the `Mapping` class, which is used to relate different ontology modules via ontology mappings.

Chapter 8

The Extended OWL Metamodel for Linking Ontologies with \mathcal{E} -Connections

Chapter 4 introduced the core of the networked ontology model, the OWL metamodel to describe single ontologies. In Chapter 6 we introduced the mapping metamodel that allows to network ontologies by defining correspondences across ontologies using mappings. In order to identify correspondences, the domains of the ontologies need to overlap. In this Chapter, we define an extension of the metamodel that allows to relate ontologies with non-overlapping (disjoint) domains using so-called link properties. We rely on the formalism of \mathcal{E} -Connections [GPS04], based on the functional-style syntax discussed in [GPS06].

The Chapter is structured in six sections: Section 8.1 provides the general motivation of the \mathcal{E} -Connections, Section 8.2 starts a general discussion on the subject of link properties and \mathcal{E} -Connections, after which Section 8.3 presents the link property entity. Next, Section 8.4 demonstrates class descriptions based on link properties and Section 8.5 presents link property axioms. Finally, Section 8.6 presents assertions.

8.1 Motivation of \mathcal{E} -Connections

Effective knowledge reuse and sharing has been a major goal of the Web Ontology Working Group. Indeed, as ontologies get larger, they become harder for reasoners to process and for humans to understand. Ontology reuse can in this case prove very useful, as it allows one to take into account existing knowledge and link it or integrate it into their knowledge base, without having to construct a large ontology from scratch. The intuition behind this is that if we think of ontologies as terminological and assertional descriptions of a certain domain, then we could split the broader ontology into different smaller ontologies that are independent and self-contained modules and that describe some more specific domain of our application.

Of course, ontology reuse requires that suitable tools should be provided for integrating and connecting the modular ontologies. The Web Ontology Language (OWL) provides some support for integrating web ontologies by defining the *owl : imports* construct, which allows one to include by reference in a knowledge base the axioms contained in another ontology, published somewhere on the Web and identified by a global name (a URI). However, this construct is not adequate enough. The most fundamental problem lies in the fact that it brings into the original ontology all the axioms of the imported one. This prohibits partial reuse and it results in exclusively syntactical modularity, as opposed to the logical modularity.

One approach that has been thoroughly studied in the last years is that of \mathcal{E} -Connections [KLWZ04]. The \mathcal{E} -Connections formalism provides a general framework for combining logical languages that are expressible as Abstract Description Systems (ADS). The main motivation and contribution is that it combines decidable logics in such a way that the resulting combined formalism remains decidable, although that can potentially come at the cost of higher worst-case complexity.

Based on this theoretical framework, the authors in [GPS06] present a syntactical and semantic extension of OWL that covers \mathcal{E} -Connections of OWL-DL ontologies, showing how such an extension can be used to

achieve modular ontology development on the Semantic Web and how \mathcal{E} -Connections provide a suitable framework for integration of Web Ontologies.

8.2 Preliminaries

In this section, we extend the definitions of the networked ontology model from Chapter 4 to take into account the \mathcal{E} -Connections formalism.

We start by taking a closer look at the \mathcal{E} -Connections. We first define the *Link Property*, which is the building stone of the \mathcal{E} -Connection.

Definition 7 A set of Link Properties ϵ_{AB} between two ontologies A and B with disjoint domains is a set of special relations between them, which relate the domain of ontology A to the domain of ontology B . A Link be conveniently conceived as a superrole relating elements of the two domains, as opposed to object properties, which relate elements of the same interpretation domain.

Based on the notion of the *Link Properties* we can build the *combined knowledge base* or \mathcal{E} -Connection of the two ontologies A and B , which talks about the domains of A and B and in addition to that about the relationships between them. [GPS04] offers a closer look at the syntax of the combined language (e.g. new available link constructors) and at the semantics by defining suitable interpretations.

8.3 The Link Property Entity

Entities are the fundamental building blocks of OWL 2 ontologies. OWL 2 has five entity types: data types, OWL classes¹, individuals, object properties and data properties. In the extended model, we also consider an additional entity, the *link property*, in line with what was discussed in the previous section.

OWL traditionally distinguishes two types of property expressions: object property expressions and data property expressions, represented by the respective metaclasses *ObjectProperty* and *DataProperty*. In the extended metamodel, we also introduce the *LinkProperty*.

A link property can possibly be defined as the inverse of an existing link property (see Example 18). The metamodel has an abstract superclass *LinkPropertyExpression* for both the usual link property and the link property defined as an inverse of another. Figure 8.1 gives its subclasses *LinkProperty* and *InverseLinkProperty*, respectively representing normal and inverse link properties. The same figure also represents the *LinkProperty* as a subclass of the more general *Entity*, as discussed previously. An inverse link property can be defined based on any, normal or inverse, link property, hence the association *inverseLinkProperty* from the metaclass *InverseLinkProperty* is connected to the superclass *LinkPropertyExpression*.

Example 18 The following example illustrates the definition of an inverse object property:
InverseLinkProperty(MadeFromIngredient)

8.4 Class Expressions

Classes group similar resources together and are the basic building blocks of class axioms. The class extension is the set of individuals belonging to the class. To define classes, OWL 2 provides next to a simple class definition (metaclass *Class*) several very expressive means for defining classes. The metamodel defines a metaclass *ClassExpression* as abstract superclass for all class definition constructs.

¹OWL provides two classes with predefined URI and semantics: owl:Thing defines the set of all objects (top concept), whereas owl:Nothing defines the empty set of objects (bottom concept).

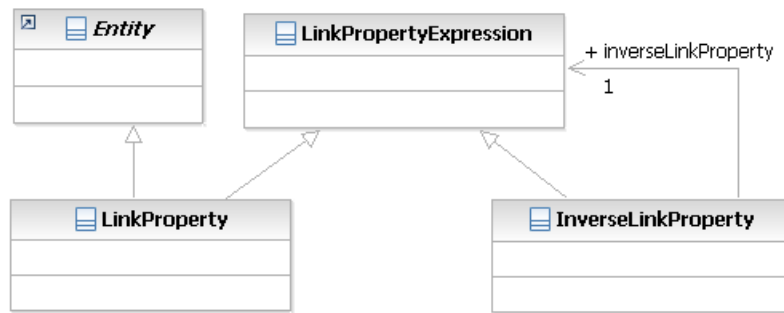


Figure 8.1: Extended metamodel: link property expressions

Link properties provide suitable constructors for building complex concept expressions, by placing different types of restrictions on link properties. This group of restrictions is actually reminiscent of the constraints placed on object properties.

Figure 8.2 gives the first group of restrictions on the property value of object properties for the context of the class.

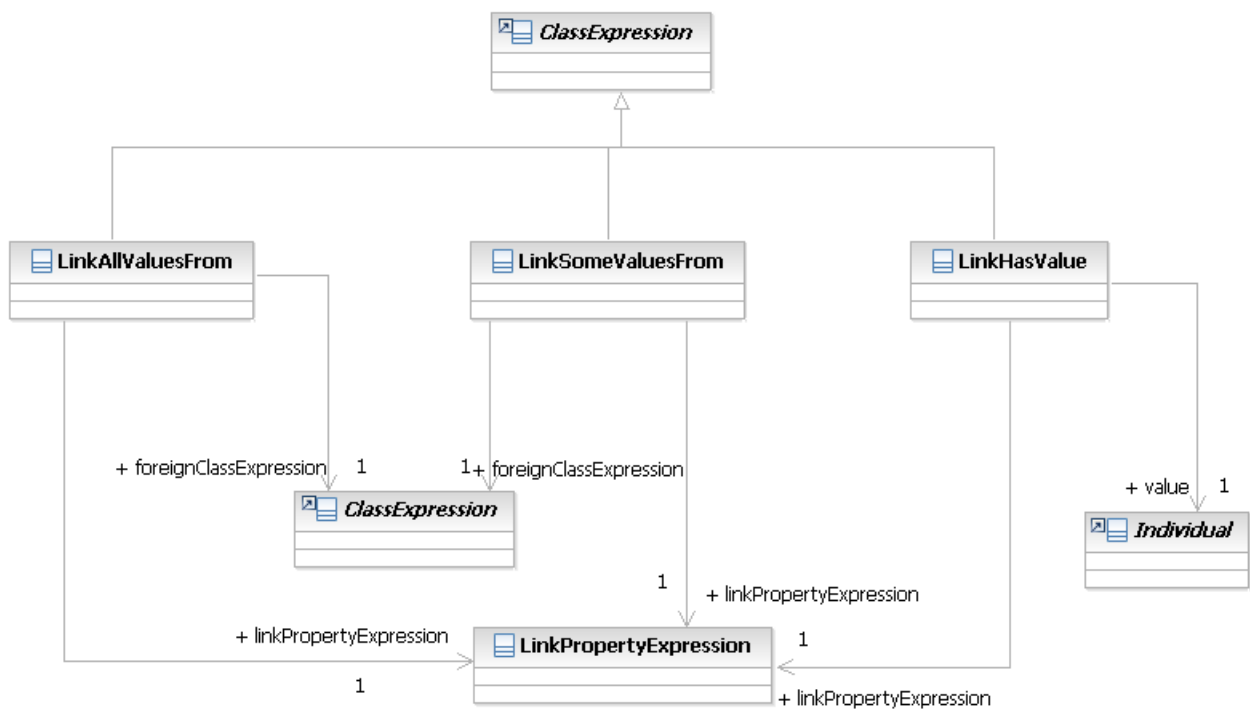


Figure 8.2: Extended metamodel: link property restrictions

The metaclass *LinkAllValuesFrom* represents the class construct that defines a class as the set of all objects which have only objects from a certain other class expression as value for a specific link property (see Example 19).

Example 19 The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description by restricting an object property:

SubClassOf(Medication ObjectAllValuesFrom(Treats Disease))

The OWL construct that defines a class as all objects which have at least one object from a certain class expression as value for a specific link property, is represented by the metaclass *LinkSomeValuesFrom*. Both *ObjectAllValuesFrom* and *ObjectSomeValuesFrom* have an association called *foreignClassExpression*, specifying the class description of the construct, whereas their association *linkPropertyExpression* specifies the link property on which the restriction is defined. The label *foreignClassExpression* describes the role of the association, namely the class expression has to come from the foreign ontology, with which the local ontology is related via the link property under consideration.² To define a class as all objects which have a certain individual as value for a specific link property, the extended model provides a construct that is represented in the metamodel by the metaclass *ObjectHasValue*, its association *linkPropertyExpression* specifying the link property, and its association *value* specifying the link property value.

The second group of link property restrictions is depicted in Figure 8.3 and it deals with restrictions on the cardinalities of link properties.

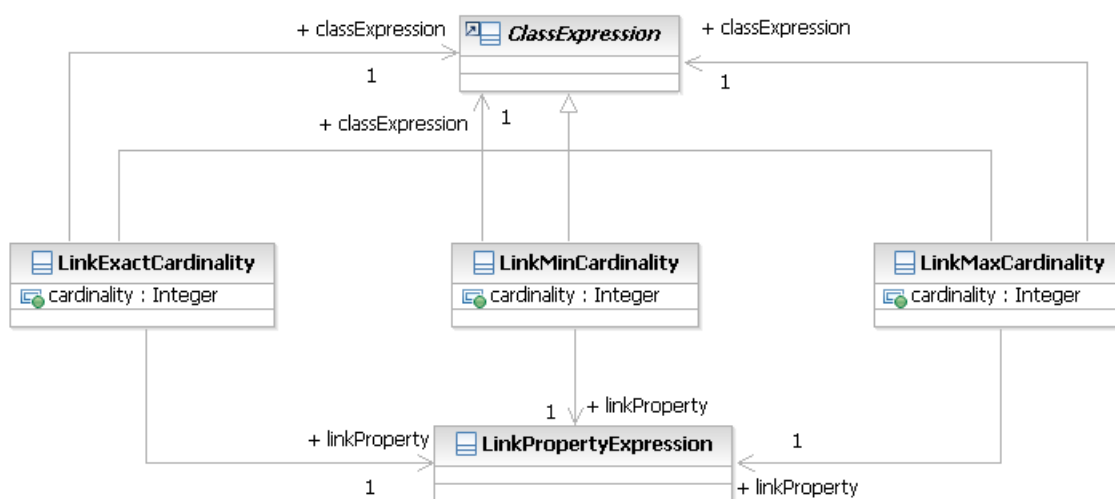


Figure 8.3: Extended metamodel: link property cardinality restrictions

A cardinality of a link property for the context of a class can be defined as a minimum, maximum or exact cardinality. The first one of this group is represented by the metaclass *LinkMinCardinality*, which defines a class of which all individuals have at least N different individuals of a certain class as values for the specified link property (N is the value of the cardinality constraint) (see Example 20).

Example 20 *The following example illustrates the definition of a class as a subclass of another class, which is defined through a class description by restricting the cardinality of an object property:*

SubClassOf(Medication LinkMinCardinality(1 madeFromIngredient Ingredient))

Secondly, the construct represented by the metaclass *LinkMaxCardinality* defines a class of which all individuals have at most N different individuals of a certain class as values for the specified link property. Finally, the construct represented by the metaclass *LinkExactCardinality* defines a class of which all individuals have exactly N different individuals of a certain class as values for the specified link property. To specify the cardinality (N) of these constructs, which is a simple integer, all three metaclasses have an attribute *cardinality*.

²This important constraint is not totally captured by the UML diagram. Although it might be somehow possible to enforce such restrictions by defining suitable OCL constraints, it would be cumbersome and thus it is preferable that tools that implement the metamodel support these constraints.

OCL constraints define that this cardinality must be a nonnegative integer³:

1. The cardinality must be nonnegative:
context LinkExactCardinality inv:
self.cardinality >= 0
2. The cardinality must be nonnegative:
context LinkMaxCardinality inv:
self.cardinality >= 0
3. The cardinality must be nonnegative:
context LinkMinCardinality inv:
self.cardinality >= 0

Additionally, they all have an association *class* and an association *linkProperty* representing the class respectively the object property involved in the statement. Note that the multiplicity of the associations called *class* have multiplicity 'zero or one' as the \mathcal{E} -Connections syntax does not define the explicit specification of the restricting class description as mandatory⁴.

8.5 Link Property Axioms

OWL defines six different kinds of axioms: entity annotations, declarations, class axioms, object property axioms, data property axioms and assertions. In the extended \mathcal{E} -Connection model we consider in addition link property axioms.

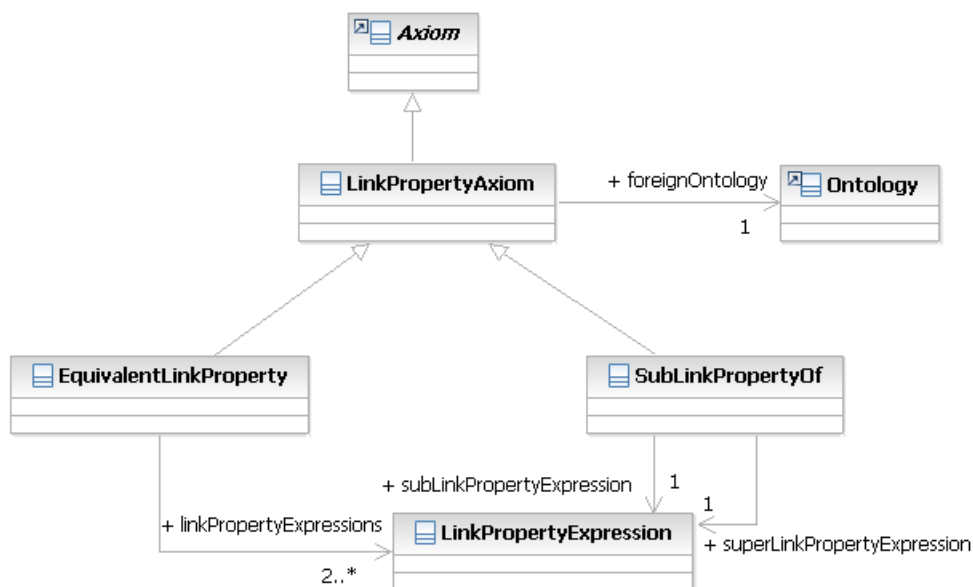


Figure 8.4: Extended metamodel: link property axioms - part 1

³Note that it would make sense to define a constraint which specifies that when a minimum and a maximum cardinality on a link property are combined to define a class, the minimum cardinality should be less than or equal to the maximum cardinality. However, as the extended do not define this restriction and rely on applications to handle this, we also do not define an OCL constraint in the metamodel.

⁴In the case where it is not explicitly defined, called an unqualified cardinality restriction, the description is owl:Thing. That is similar as with the

The group of link property axioms contains constructs defining relations between different link properties, definitions of domain and range and specifications of property characteristics. We first introduce the various relations between link properties in Figure 8.4.

The first one, represented by the metaclass *SubLinkPropertyOf*, defines that the property extension of one link property, specified through the association *subLinkPropertyExpression*, is a subset of the extension of another link property, specified through the association *superLinkPropertyExpression*. The metaclass *EquivalentLinkProperty* represents a construct that defines that the property extensions of two or more link properties are the same.

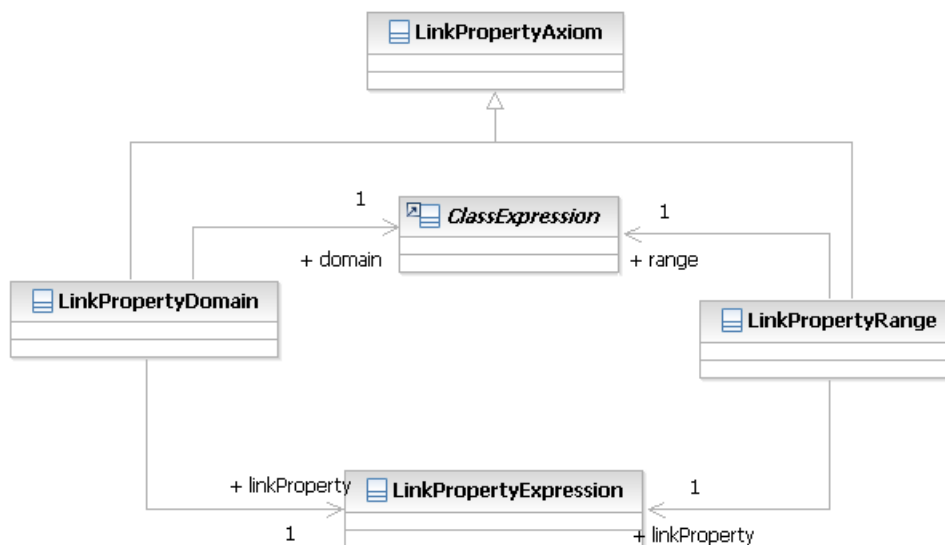


Figure 8.5: Extended metamodel: link property axioms - part 3

To define the class to which the subjects of a link property belong, the extended model provides the domain concept (see Example 21), whereas a range specifies the class to which the objects of the property, the property values, belong. Figure 8.5 presents the metamodel elements for the constructs defining a link property domain and range.

Example 21 *The following example illustrates the definition of the domain of an object property:*
LinkPropertyDomain(madeFromIngredient Medication)

The metaclass *LinkPropertyDomain* specifies a link property and its domain via the associations *property* respectively *domain*. Similarly, the metaclass *LinkPropertyRange* represents the construct to define the range of a link property.

The remaining OWL link property axioms take a link property and assert a characteristic to it. In doing so, link properties can be defined to be functional or inverse functional.

A functional link property is a property for which each subject from the domain, can have only one value in the range, whereas an inverse functional link property can have only one subject in the domain for each value in the range.

Figure 8.6 demonstrates that each of these two axioms has an own metaclass with an association *linkProperty* to the class *LinkPropertyExpression*, specifying the property on which the characteristic is defined.

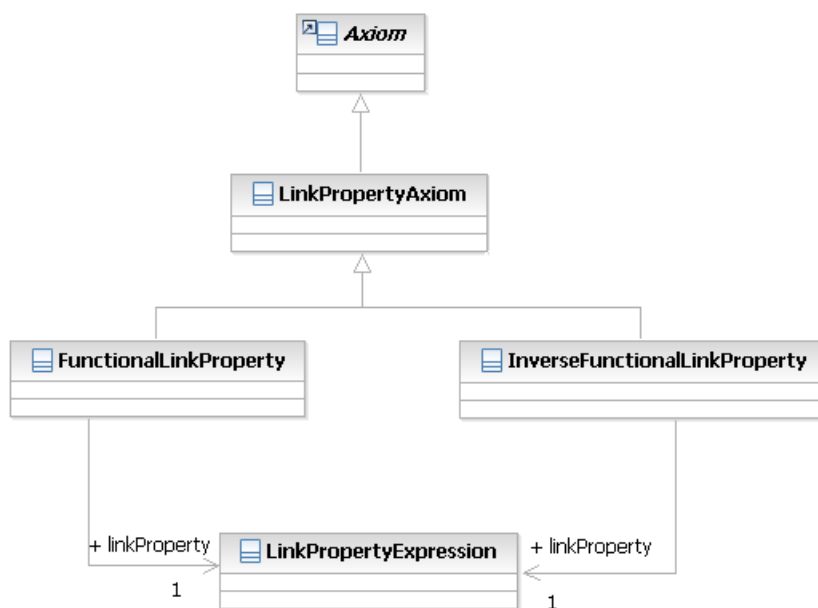


Figure 8.6: Extended metamodel: link property axioms - part 2

8.6 Assertions

In addition to the existing OWL assertions, the extended model defines a further assertion axiom to state assertions about link properties.

To specify the value of a specified individual under a certain link property, the extended model provides the link property assertion construct (see Example 22).

Example 22 *The following example illustrates how the value of an individual under a link property, is defined: `LinkPropertyAssertion(madeFromIngredient aspirin acetylsalicylicacid)`*

Figure 8.7 depicts the construct in the metamodel as the metaclass *LinkPropertyAssertion*. It has an association to *property* representing the involved link property, and two associations to *Individual* representing the subject and the object of the assertion.

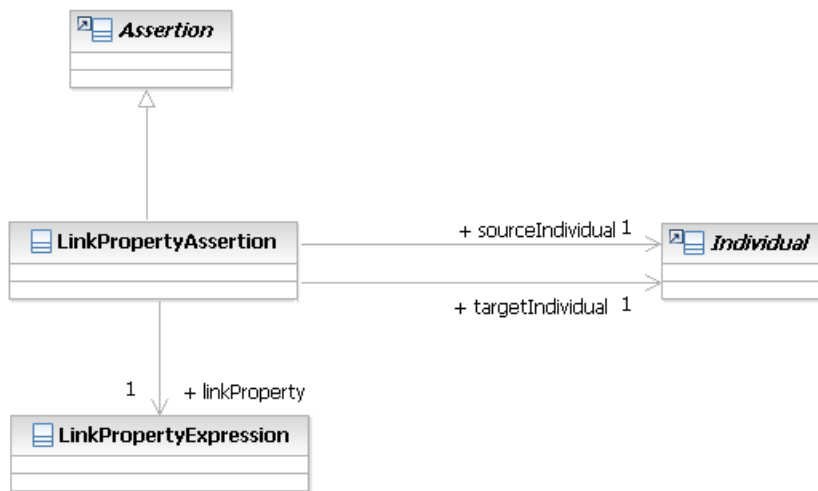


Figure 8.7: Extended metamodel: assertions

Part III

Metadata for Networked Ontologies

Chapter 9

The NeOn Ontology Metadata Vocabulary

Ontologies have undergone an enormous development and have been applied in many domains within the last years, especially in the context of the Semantic Web. Currently however, efficient knowledge sharing and reuse, a pre-requisite for the realization of the Semantic Web vision, is a difficult task. It is hard to find and share existing ontologies because of the lack of standards for documenting and annotating ontologies with metadata information. Without ontology-specific metadata, developers are not able to reuse existing ontologies, which leads to problems of interoperability as well as duplicate efforts. Then, in order to provide a basis for an effective access and exchange of ontologies across the web, it is necessary to agree on a standard for ontology metadata. This standard then provides a common set of terms and definitions describing ontologies and is called metadata vocabulary.

In the remainder of this chapter we present a compact overview of our contribution to the alleviation of this situation: the ontology metadata standard OMV (**O**ntology **M**etadata **V**ocabulary), which specifies reusability-enhancing ontology features for human and machine processing purposes.

Overview of the latest changes in OMV

- OMV-overview Figure updated
- dataProperty added: reference
- Changed the cardinality of statistics properties to 0:1 (optional)
- Moved the following dataProperties to DL extension: containsABox, containsRBox, containsTBox, expressiveness
- Changed cardinality of objectProperties isOfType, hasOntologySyntax to 0:1 (optional)
- Added predefined instances of OntologyEngineeringTool
- Fixed developesxxx to developsxxx
- Changed objectProperty usedKnowledgeRepresentationParadigm to conformsToKnowledgeRepresentationParadigm
- Moved dataProperty isConsistentAccordingToReasoner to evaluation extension
- Added objectProperty endorses
- Deleted objectProperties: developsOntologyEngineeringMethodology, developsOntologyEngineeringTool, developsOntologyLanguage, developsOntologySyntax, specifiesLicenseModel, specifiesKnowledgeRepresentationParadigm, definesOntologyType

- Added objectProperties develops, specifies, defines
- Updated inverse relations
- Minor typos fixed
- Updated description in section "Identification, Versioning and Location" to OWL 2 specification.

9.1 Preliminary considerations

9.1.1 Metadata Categories

OMV differentiates among the following three occurrence constraints for metadata elements—according to their impact on the prospected reusability of the described ontological content:

- **Required** – mandatory metadata elements. Any missing entry in this category leads to an incomplete description of the ontology.
- **Optional** – important metadata facts, but not strongly required.
- **Extensional** – specialized metadata entities, which are not considered to be part of the core metadata scheme.

Complementary to this classification we organize the metadata elements, according to the type and purpose of the contained information, as follows:

- **General** – elements providing general information about the ontology.
- **Availability** – information about the location of the ontology (e.g. its URI or URL where the ontology is published on the Web)
- **Applicability** – information about the intended usage or scope of the ontology.
- **Format** – information about the physical representation of the resource. In terms of ontologies, these elements include information about the representation language(s) in which the ontology is formalized.
- **Provenance** – information about the organizations contributing to the creation of the ontology.
- **Relationship** – information about relationships to other resources. This category includes versioning, as well as conceptual relationships such as extensions, generalization/specialization and imports.
- **Statistics** - various metrics on the underlying graph topology of an ontology (e.g. number of classes)
- **Other** - information not covered in the categories listed above.

Note that the introduced classification dimensions are not intended to be part of the metadata scheme itself, but will be taken into consideration by the implementation of several metadata support facilities. The first dimension is relevant for a metadata creation service in order to ensure a minimal set of useful metadata entries for each of the described resources. The second can be used in various settings mainly to reduce the user-perceived complexity of the metadata scheme whose elements can be structured according to the corresponding classes.

9.2 Ontology Metadata Requirements

We elaborated an inventory of requirements for the metadata model as a result of a systematic survey of the state of the art in the area of ontology reuse. Besides a scientific analysis, we conducted extensive literature research, which focused on theoretical methods [PM01, GPS99, LTGP04], but also on case studies on reusing existing ontologies [UHW⁺98, RVMS99, PBMT05], in order to identify real-world needs of the community w.r.t. a descriptive metadata format for ontologies. Further on, the requirements analysis phase was complemented by a comparative study of existing (ontology-independent) metadata models and of tools such as ontology repositories and libraries (implicitly) making use of metadata-like information. Several aspects are similar to other metadata standards such as Dublin Core. Differences arise however if we consider the semantic nature of ontologies, which are much more than plain Web information sources. In accordance to one of the major principles in Ontological Engineering, an ontology comprises a conceptual model of a particular domain of interest, represented at the knowledge level, and multiple implementations using knowledge representation languages. These two components are characterized by different properties, can be developed and maintained separately. The main requirements identified in this process step are the following:

Accessibility: Metadata should be accessible and processable for both humans and machines. While the human-driven aspects are ensured by the usage of natural language concept names, the machine-readability requirement can be implemented by the usage of Web-compatible representation languages (such as XML or Semantic Web languages, see below).

Usability: It is important to build a metadata model which 1) reflects the needs of the majority of ontology users—as reported by current case studies in ontology reuse—but at the same time 2) allows proprietary extensions and refinements in particular application scenarios. From a content perspective, usability can be maximized by taking into account multiple metadata types, which correspond to specific viewpoints on the ontological resources and are applied in various application tasks. Despite the broad understanding of the metadata concept and the use cases associated to each definition, several key aspects of metadata information have already been established across computer science fields [Org04]:

- **Structural metadata** relates to statistical measures on the graph structure which underlies an ontology. In particular we mention the number of specific ontological primitives (e.g. number of classes or instances). The availability of structural metadata influences the usability of an ontology in a concrete application scenario, as size and structure parameters constraint the type of tools and methods which can be applied to aid the reuse process.
- **Descriptive metadata** relates to the domain modeled in the ontology in form of keywords, topic classifications, textual descriptions of the ontology contents etc. This type of metadata plays a crucial role in the selection of appropriate reuse candidates, a process which includes requirements w.r.t. the domain of the ontologies to be re-used.
- **Administrative metadata** provides information to help manage ontologies, such as when and how it was created, rights management, file format and other technical information.

Interoperability: Similarly to the ontology it describes, metadata information should be available in a form which facilitates metadata exchange among applications. While the syntactical aspects of interoperability are covered by the usage of standard representation languages (see “Accessibility”), the semantical interoperability among machines handling ontology metadata information can be ensured by means of an formal and explicit representation of the meaning of the metadata entities, i.e. by conceptualizing the metadata vocabulary itself as an ontology.

9.3 Ontology Metadata Vocabulary

This section gives an overview of the core design principles applied for the realization of the OMV metadata scheme, which is described in detail later in this section.

9.3.1 Core and Extensions

Following the usability constraints identified during the requirements analysis, we decided to design the OMV scheme modularly; OMV distinguishes between the OMV Core and various OMV Extensions. The former captures information which is expected to be relevant to the majority of ontology reuse settings. However, in order to allow ontology developers and users to specify task- or application-specific ontology-related information we foresee the development of OMV extension modules, which are physically separated from the core scheme, while remaining compatible to its elements.

9.3.2 Ontological representation

Due to the high accessibility and interoperability requirements, as well as the nature of the metadata, which is intended to describe Semantic Web ontologies, the conceptual model designed in the previous step was implemented in the OWL language. An implementation as XML-Schema or DTD was estimated to restrict the functionality of the ontology management tools using the metadata information (mainly in terms of retrieval capabilities) and to impede metadata exchange at semantical level. Further on, a language such as RDFS does not provide a means to distinguish between required and optional metadata properties. The implementation was performed manually by means of a common ontology editor.

9.3.3 Identification, Versioning and Location

An important issue that has to be addressed when describing ontologies is the ability to identify and manage multiple versions and physical representations of one ontology. The OWL ontology language itself did not provide the means to address this issue until the latest release of OWL 2¹ where they partially address it: In OWL 2, an ontology is identified by an IRI (if available). Hence, OWL 2 extends OWL 1, which uses Uniform Resource Identifiers (URIs) [BLFM98]. Additionally, an ontology may have a version IRI, which is used to identify the version of the ontology. The version IRI may, but need not be equal to the ontology IRI. However, the specification provides no mechanism for enforcing these constraints across the entire Web.

Versioning In general, a version is a variant of an ontology that is usually created after applying changes to an existing variant. Therefore we need a way to unambiguously identify the different versions as well as to keep track of the relationships between them. Based on [KF01], we consider that changes in ontologies are caused by: (i) changes in the domain; (ii) changes in the shared conceptualization; (iii) changes in the specification. Taking the definition of an ontology as a specification of a conceptualization, (i) and (ii) are semantic changes that lead to the creation of a new conceptualization, while (iii) is just a change in the representation of the same conceptualization (also known as a new revision) (e.g. updates of natural language descriptions of ontology elements). In any case, the change(s) result in a different physical representation of the ontology (i.e. different version). Consequently, it should be possible to identify each of those versions.

However, as we mentioned above OWL 1 does not distinguish between the notion of an ontology and a version of an ontology at all. It may thus be that different versions of ontology carry the same logical URI. Only in the latest specification of OWL 2, it is proposed that the ontology IRI and the version IRI together identify a particular version from an ontology series i.e. the set of all the versions of a particular ontology identified using a common ontology IRI. In each ontology series, exactly one ontology version is regarded as the current one.

¹<http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/>

So, typically (as OWL 2 has just been released) existing ontologies are identified by an URI, which according to [BLFM98] is a compact string of characters for identifying an abstract or physical resource. In [KF01] the authors propose that any version that constitutes a new conceptualization (i.e. changes of type (i) and (ii)) should have a unique URI, however in practice different versions of same ontology might share the same URI. Furthermore, even if a revision constitutes the same conceptualization of an ontology it is physically represented in a different file which might have additional metadata (e.g. updated ontology description, descriptions in different natural languages, different file location, etc.).

In OMV we describe a particular representation of an ontology, i.e. an ontology in a particular version at a particular physical location. That means that every different version of an ontology has a different OMV related metadata.

Currently however, most ontologies either do not provide any version information at all or the ontology editors explicitly do not want to change the version of the ontology after making some changes. In those cases, whenever the ontology changes, the related OMV annotation will have to be updated accordingly instead of creating a new OMV instance (i.e. including updating the date of the last time the ontology was modified).

Resource Location In addition to the issue of versioning, an ontology (or a version of an ontology) can be located at different locations. Thus, ontologies with the same logical URI (IRI in OWL 2) may exist at different physical locations, possibly even with different content. Similar to versioning, only the latest OWL 2 specification provides some conventions for the location of an ontology: Each ontology is associated with an ontology document, which physically contains the ontology stored in a particular way. The document containing the current version of an ontology series with some IRI OI should be accessible from OI. A particular version of OI with version IRI VI, should be accessible from VI.

In practice however (as OWL 2 has just been released), the location of the ontology is not necessarily equal to the URI of the ontology. So, following the approach for versioning, we rely on a composite identifier consisting of the logical identifier i.e. URI (ontology IRI in OWL 2) plus optional version identifier (version IRI in OWL 2), and a resource locator that specifies the actual physical location.

Of course, the optional version identifier and the optional resource locator can be combined, such that we end up with a tripartite identifier (URI, version, resource locator).

Illustrative Example In order to clarify the previous discussion consider the following scenario: Initially, we have the first implementation of ontology OWLODM (i.e. <http://owlodm.ontoware.org/OWL1.0>) which provides a metamodel for the ontology language OWL 1.0. A fragment of the OMV description for OWLODM version 1.0 is the following:

```
<omv:Ontology rdf:about="&j;OWL1.0v1.0">
  <omv:URI rdf:datatype="&xsd:string">http://owlodm.ontoware.org/OWL1.0</omv:URI>
  <omv:version rdf:datatype="&xsd:string">1.0</omv:version>
  <omv:resourceLocator rdf:datatype="&xsd:string">
    http://ontoware.org/frs/download.php/307/owl10.owl</omv:resourceLocator>
  <omv:acronym rdf:datatype="&xsd:string">OWLODM</omv:acronym>
  <omv:description rdf:datatype="&xsd:string">OWL Object Definition Metamodel
    (ODM) allows interoperability of OWL ontologies with MOF-compatible
    software environments</omv:description>
  <omv:name rdf:datatype="&xsd:string">OWL Ontology Definition Metamodel</omv:name>
  <omv:numberOfClasses rdf:datatype="&xsd:unsignedInt">35</omv:numberOfClasses>
  <omv:numberOfProperties rdf:datatype="&xsd:unsignedInt">22</omv:numberOfProperties>
  <omv:hasCreator rdf:resource="#PeterHaase"/>
  <omv:hasDomain rdf:resource="&c;Knowledge Representation"/>
  <omv:creationDate rdf:datatype="&xsd:string">2007-02-12</omv:creationDate>
  ...
</omv:Ontology>
```

A change in the domain modelled by OWLODM (i.e. the definition of OWL 1.1) was reflected in a new *version* of the OWLODM ontology, namely version 1.1. This change led to a semantic change of the ontology (i.e. change of type (i)), and therefore a new URI was defined for OWLODM (i.e. <http://owlodm.ontoware.org/OWL1.1>). A fragment of the OMV description for OWLODM version 1.1 is the following:

```
<omv:Ontology rdf:about="&j;OWL1.1v1.1">
  <omv:URI rdf:datatype="&xsd:string">http://owlodm.ontoware.org/OWL1.1</omv:URI>
  <omv:version rdf:datatype="&xsd:string">1.1</omv:version>
  <omv:resourceLocator rdf:datatype="&xsd:string">
```

```

http://ontoware.org/frs/download.php/365/owl11.owl</omv:resourceLocator>
<omv:acronym rdf:datatype="xsd:string">OWLODM</omv:acronym>
<omv:description rdf:datatype="xsd:string">OWL Object Definition Metamodel
(ODM) allows interoperability of OWL ontologies with MOF-compatible
software environments</omv:description>
<omv:name rdf:datatype="xsd:string">OWL Ontology Definition Metamodel</omv:name>
<omv:numberOfClasses rdf:datatype="xsd:unsignedInt">76</omv:numberOfClasses>
<omv:numberOfProperties rdf:datatype="xsd:unsignedInt">35</omv:numberOfProperties>
<omv:hasCreator rdf:resource="#PeterHaase"/>
<omv:hasDomain rdf:resource="c:Knowledge Representation"/>
<omv:creationDate rdf:datatype="xsd:string">2007-08-09</omv:creationDate>
...
</omv:Ontology>

```

Finally, a new version of OWLODM (i.e. version 1.2) was released as a result of a refinement. In this case, the change was at the level of the specification of the ontology (i.e. change type (iii)), in particular the renaming of a property and hence the URI was not updated. A fragment of the OMV description for the OWLODM version 1.2 is the following:

```

<omv:Ontology rdf:about="j;OWL1.1v1.2">
<omv:URI rdf:datatype="xsd:string">http://owlodm.ontoware.org/OWL1.1</omv:URI>
<omv:version rdf:datatype="xsd:string">1.2</omv:version>
<omv:resourceLocator rdf:datatype="xsd:string">
http://ontoware.org/frs/download.php/366/owl11.owl</omv:resourceLocator>
<omv:acronym rdf:datatype="xsd:string">OWLODM</omv:acronym>
<omv:description rdf:datatype="xsd:string">OWL Object Definition Metamodel
(ODM) allows interoperability of OWL ontologies with MOF-compatible
software environments</omv:description>
<omv:name rdf:datatype="xsd:string">OWL Ontology Definition Metamodel</omv:name>
<omv:numberOfClasses rdf:datatype="xsd:unsignedInt">76</omv:numberOfClasses>
<omv:numberOfProperties rdf:datatype="xsd:unsignedInt">35</omv:numberOfProperties>
<omv:hasCreator rdf:resource="#PeterHaase"/>
<omv:hasDomain rdf:resource="c:Knowledge Representation"/>
<omv:creationDate rdf:datatype="xsd:string">2007-08-10</omv:creationDate>
...
</omv:Ontology>

```

As we can see from the previous simple example, the URI is not enough to identify individually each version of the ontology. Besides, in practice not every semantic change leads to the definition of a new URI (as in this example). Even more, in this example it was enough to use the URI plus the version to identify each physical implementation, however it could also be possible that the same ontology version is located at two (or more) different physical location, where each of them could have even different content as we anticipated in the previous section. In that case we will also need to use the location of the ontology to identify a particular implementation (i.e. URI plus version plus location).

9.3.4 OMV core metadata entities

The main classes and properties of the OMV ontology are illustrated in Figure 9.1.²

Additionally to the main class `Ontology` the metadata scheme contains further elements describing various aspects related to the creation, management and usage of an ontology. We will briefly discuss these in the following. In a typical ontology engineering process `Persons` or `Organisation(s)` are developing ontologies. We group these two classes under the generic class `Party` by a subclass-of relation. A `Party` can have several locations by referring to a `Location` individual and can *create*, *contribute* to ontological resources i.e. `Ontology Implementations`. Review details and further information can be captured in an extensional OMV module (see [HP05]). Further on we provide information about the engineering process the ontology originally resulted from in terms of the classes `OntologyEngineeringMethodology`, `OntologyEngineeringTool` and the attributes `version`, `status`, `creationDate` and `modificationDate`. Again these can be elaborated as an extension of the core metadata scheme. The usage history of the ontology is modelled by classes such as the `OntologyTask` and `LicenceModel`. The scheme also contains a representation of the most significant intrinsic features of an ontology. Details on ontology languages are representable with the help of the classes

²Please notice, that not all classes and properties are included. The ontology is available for download in several ontology formats at <http://omv.ontoware.org/>

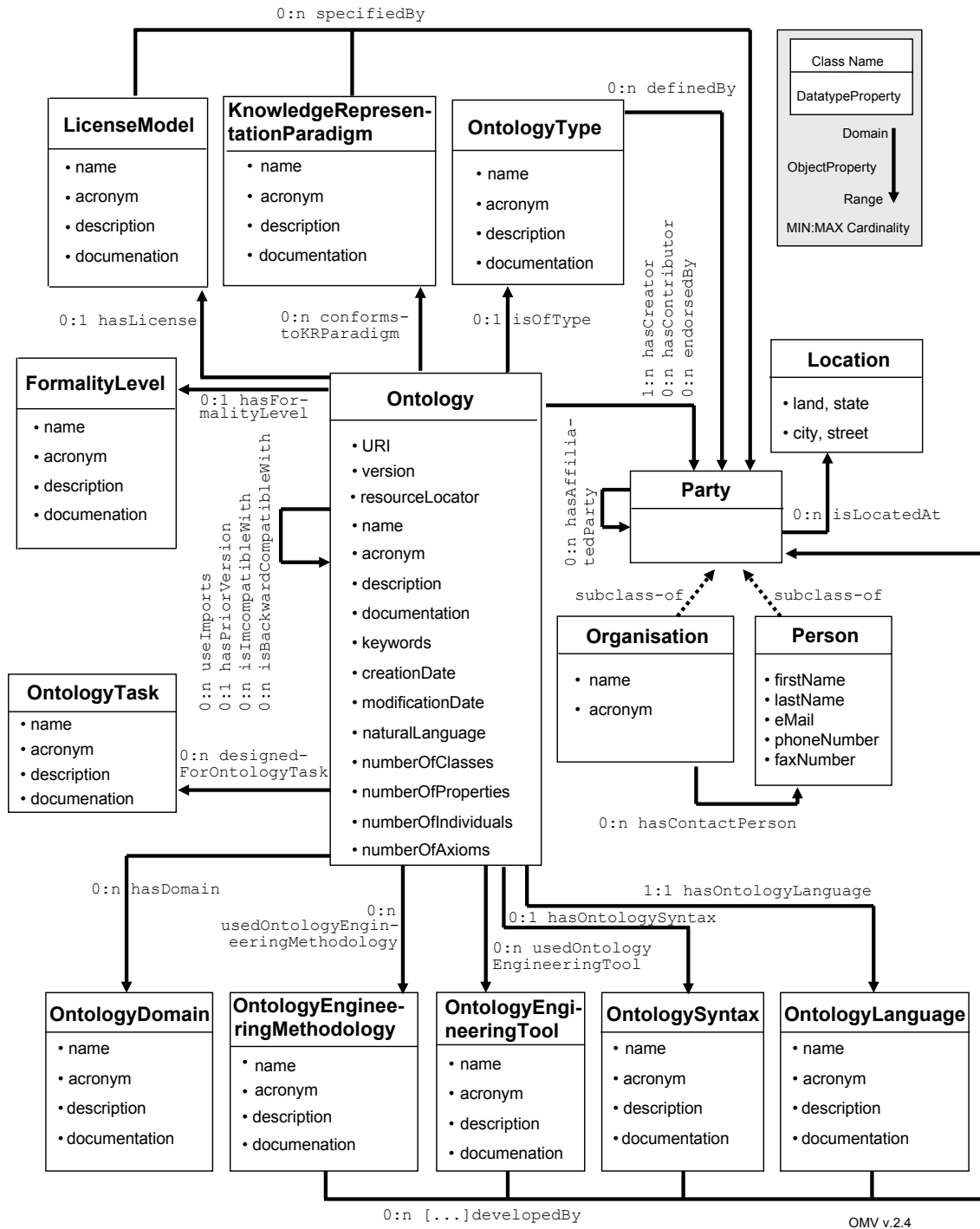


Figure 9.1: OMV overview

OntologySyntax, OntologyLanguage and KnowledgeRepresentationParadigm. Ontologies might be categorized along a multitude of dimensions. One of the most popular classification differentiates among application, domain, core, task and upper-level ontologies. A further classification relies on their level of formality and types of Knowledge Representation (KR) primitives supported, introducing catalogues, glossaries, thesauri, taxonomies, frames etc. as types of ontologies. These can be modeled as instances of the class `OntologyType`, while generic formality levels are introduced with the help of the class `FormalityLevel`. The domain the ontology describes is represented by the class `OntologyDomain` referencing a pre-defined topic hierarchy such as the DMOZ hierarchy. Further content information can be provided as values of the `DatatypeProperties` `description`, `keywords`, and `documentation`. Finally the metadata scheme gives an overview of the graph topology of an `Ontology` with the help of several graph-related metrics represented as integer values of the `DatatypeProperties` `numberOfClasses`, `numberOfProperties`, `numberOfAxioms`, `numberOfIndividuals`.

In the appendix of this deliverable we provide a detailed description of the OMV core model. For additional information please refer to [HP05].

Chapter 10

Conclusion

10.1 Summary

Next generation semantic applications will be characterized by a large number of networked ontologies; as a consequence the complexity of semantic applications increases. In the NeOn project we address this challenge by creating an open infrastructure, and associated methodology, to support the development life-cycle of such a new generation of semantic applications. In this deliverable we have presented an updated version of our metamodel for the specification of networked ontologies that will serve as a foundation for this infrastructure. We have followed the metamodeling approach of Model Driven Architectures for the specification of the metamodel. Specifically, we have presented four modules of the metamodel:

1. The OWL 2 metamodel serves as the core of our networked ontology model,
2. the rule metamodel covering both SWRL and F-Logic,
3. the mapping metamodel allows for the specification of ontology mappings that describe correspondences between ontology elements in a network of ontologies, and
4. the metamodel for modular ontologies builds on the OWL and mapping metamodel to model ontologies as reusable components in a network of ontologies.

Further, in this deliverable we have presented the current version of OMV as an Ontology Metadata Vocabulary, which aims at facilitating the task of sharing, searching, reusing and managing the relationships between ontologies.

We expect that the extended models presented in this deliverable will co-exists with our initial models presented in D1.1.1 and D1.1.2 in the tool infrastructure since to a large extend our new models complement our initial models. In particular, we expect that the OWL 1 metamodel will co-exist with our OWL 2 metamodel until OWL 2 is accepted as a standard and fully supported in the tool infrastructure.

Appendix A

Naming Conventions

Choosing a naming convention for ontology modeling and adhere to these conventions makes the ontology easier to understand and helps to avoid some common modeling mistakes. Therefore, for the modeling of ontologies in the context of the NeOn project, we adopted the following set of conventions for ontology elements (i.e. classes, properties and instances)

A.1 Delimiters and capitalization

- **Class Names** - Class names are capitalized. If the class name contains more than one word, we use concatenated words and capitalize each new word. I.e. "Ontology" "OntologySyntax"
- **Property Names** - Property names use lower case. If the property name contains more than one word, we use concatenated words where the first word is all in lower case and capitalize each subsequent new word. I.e. "name" "naturalLanguage" "hasLicense"
- **Instance Names** - Instance names use lower case. If the instance name contains more than one word, we use concatenated words where the first word is all in lower case and capitalize each subsequent new word. I.e. "peter".

A.2 Property naming

The use of property naming conventions helps to identify in an easy way ontology elements. NeOn uses specific prefixes to identify **DatatypeProperty** and **ObjectProperty**. Thus, the ObjectProperties start with a verb specifying how the two classes are related to each other. I.e. "specifiedBy" "usedOntologyEngineeringTool" "hasOntologySyntax".

A.3 Singular form

The convention adopted by NeOn was to use names for classes, properties and instances in singular form. The decision was based on the fact that singular form is used more often in practice in many domains. Besides, when working with XML, for example, importing legacy XML or generating XML feeds from the ontology, it is necessary to make sure to use a singular form since this is expected convention for XML tags.

A.4 Additional considerations

- When a word within a name is all capitals, the next word should start in lower case. An hypothetical example: "URLoriginal"

- Do not add strings such as "class" or "attribute", and so on to the names.
- Do not concatenate the name of the class to the properties or instances, i.e. there is no "ontology-Name" "ontologySyantxName"
- Try to avoid abbreviations on the names of the ontology elements. The use of abbreviations in the names can lead to names difficult to understand, therefore unless the names are self explanatory, we will avoid them.

Appendix B

OMV Core Ontology

In the following we introduce the metadata elements of OMV, the first metadata standard for ontologies. As aforementioned, OMV is formalized as an OWL ontology. A metadata element is modelled either by means of classes and individuals or by means of valued properties. The decision for one of these two alternatives was justified by the complexity of the corresponding metadata element. If the value/content of a metadata element can be easily mapped to conventional data types (numerical, literal, list values) the metadata element is usually represented as a `DatatypeProperty`. Complex metadata elements which do not fall into the previous category are modelled by means of additional classes linked by `ObjectProperties`.

The description of the model is grouped along the core classes of the ontology. For each class we describe the meaning of its properties and additional usage and occurrence constraints.

B.1 Ontology

Aspects of specific realizations are covered modular (and extendable) by the class `Ontology`.

Ontology	
Name Type Identifier	Ontology class
Definition	An implementation of a conceptual model
OMV version	0.1
Comments	None

Table B.1: Class: Ontology

URI	
Name Type Identifier	URI DatatypeProperty
Occurrence Constraint Category	required General information
Definition	The URI of the ontology which is described by this metadata. It serves as a logical identifier and is not necessarily the physical location
Domain Range Cardinality	omv:Ontology xsd:string 1:1
OMV version	0.1
Comments	None

Table B.2: Property: URI

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required General information
Definition	The name by which an ontology is formally known
Domain Range Cardinality	omv:Ontology xsd:string 1:n
OMV version	0.1
Comments	The ontology can have many names (e.g. names in different languages)

Table B.3: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	A short name by which an ontology is formally known
Domain Range Cardinality	omv:Ontology xsd:string 1:1
OMV version	0.1
Comments	None

Table B.4: Property: acronym

description	
Name	description
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	General information
Definition	Free text description of an ontology
Domain	omv:Ontology
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.5: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	General information
Definition	URL for further documentation
Domain	omv:Ontology
Range	xsd:string
Cardinality	0:1
OMV version	0.2
Comments	None

Table B.6: documentation

reference	
Name	reference
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	General information
Definition	List of bibliographic references describing the ontology and its applications
Domain	omv:Ontology
Range	xsd:string
Cardinality	0:1
OMV version	2.4
Comments	None

Table B.7: reference

notes	
Name	notes
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	General information
Definition	Additional information about the ontology that is not included somewhere else (e.g. information that you do not want to include in the documentation)
Domain	omv:Ontology
Range	xsd:string
Cardinality	0:1
OMV version	2.2
Comments	None

Table B.8: notes

naturalLanguage	
Name Type Identifier	naturalLanguage DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	The language of the content of the ontology, i.e. English, German, etc.
Domain Range Cardinality	omv:Ontology xsd:string 0:n
OMV version	0.1
Comments	Pre-defined values according to the names of languages defined in ISO 639

Table B.9: Property: naturalLanguage

keywords	
Name Type Identifier	keywords DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	List of keywords related to an ontology
Domain Range Cardinality	omv:Ontology xsd:string 0:n
OMV version	0.1
Comments	Typically this set includes words that describe the content of the ontology

Table B.10: Property: keywords

keyClasses	
Name Type Identifier	keyClasses DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	Representative classes in the ontology
Domain Range Cardinality	omv:Ontology xsd:string 0:n
OMV version	2.2
Comments	none

Table B.11: Property: keyClasses

status	
Name Type Identifier	status DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	The tracking information for the contents of the ontology
Domain Range Cardinality	omv:Ontology xsd:string 0:1
OMV version	0.1
Comments	Pre-defined values

Table B.12: Property: status

creationDate	
Name Type Identifier	creationDate DatatypeProperty
Occurrence Constraint Category	required General information
Definition	Date when the ontology was initially created.
Domain Range Cardinality	omv:Ontology xsd:date 1:1
OMV version	0.1
Comments	In case versioning information is being used, it refers to the date of creation of this particular version. Otherwise it refers to the date of creation of the first version of this ontology

Table B.13: Property: creationDate

modificationDate	
Name Type Identifier	modifiedDate DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	Date of the last modification made to the ontology.
Domain Range Cardinality	omv:Ontology xsd:date 0:1
OMV version	0.1
Comments	In case versioning information is being used, it is not applicable, otherwise it refers to the date of last modification of this ontology

Table B.14: Property: modificationDate

hasContributor	
Name	hasContributor
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	Contributors to the creation of the ontology
Domain	omv:Ontology
Range	omv:Party
Cardinality	0:n
OMV version	0.1
Comments	None

Table B.15: Property: hasContributor

hasCreator	
Name	hasCreator
Type	ObjectProperty
Identifier	
Occurrence Constraint	required
Category	Provenance information
Definition	Main responsible for the creation of the ontology
Domain	omv:Ontology
Range	omv:Party
Cardinality	1:n
OMV version	0.1
Comments	None

Table B.16: Property: hasCreator

usedOntologyEngineeringTool	
Name	usedOntologyEngineeringTool
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	Information about the tool used to create the ontology
Domain	omv:Ontology
Range	omv:OntologyEngineeringTool
Cardinality	0:n
OMV version	0.1
Comments	See section B.5

Table B.17: Property: usedOntologyEngineeringTool

usedOntologyEngineeringMethodology	
Name	usedOntologyEngineeringMethodology
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	Information about the method model used to create the ontology
Domain	omv:Ontology
Range	omv:OntologyEngineeringMethodology
Cardinality	0:n
OMV version	0.1
Comments	See section B.4

Table B.18: Property: usedOntologyEngineeringMethodology

conformsToKnowledgeRepresentationParadigm	
Name Type Identifier	conformsToKnowledgeRepresentationParadigm ObjectProperty
Occurrence Constraint Category	optional Provenance information
Definition	Information about the paradigm model used to create the ontology
Domain Range Cardinality	omv:Ontology omv:KnowledgeRepresentationParadigm 0:n
OMV version	0.1
Comments	See section B.8

Table B.19: Property: conformsToKnowledgeRepresentationParadigm

endorsedBy	
Name Type Identifier	endorsedBy ObjectProperty, inverseOf(endorses)
Occurrence Constraint Category	optional Provenance information
Definition	The parties (i.e. organisations, people) that have expressed support or approval to this ontology.
Domain Range Cardinality	omv:Ontology omv:Party 0:n
OMV version	0.1
Comments	None

Table B.20: Property: endorsedBy

hasDomain	
Name Type Identifier	hasDomain ObjectProperty
Occurrence Constraint Category	optional Applicability information
Definition	The subject domain of the ontology
Domain Range Cardinality	omv:Ontology omv:OntologyDomain 0:n
OMV version	0.8
Comments	Typically, the domain can refer to established topic hierarchies such as the general purpose topic hierarchy DMOZ or the domain specific topic hierarchy ACM for the computer science domain. See section B.11

Table B.21: Property: hasDomain

isOfType	
Name Type Identifier	isOfType ObjectProperty
Occurrence Constraint Category	optional Applicability information
Definition	The nature of the content of the ontology
Domain Range Cardinality	omv:Ontology omv:OntologyType 0:1
OMV version	0.1
Comments	Pre-defined values. See section B.2

Table B.22: Property: isOfType

designedForOntologyTask	
Name Type Identifier	designedForOntologyTask ObjectProperty
Occurrence Constraint Category	optional Applicability information
Definition	The purpose for which the ontology was originally designed
Domain Range Cardinality	omv:Ontology omv:OntologyTask 0:n
OMV version	0.9
Comments	See section B.10

Table B.23: Property: designedForOntologyTask

hasFormalityLevel	
Name Type Identifier	hasFormalityLevel ObjectProperty
Occurrence Constraint Category	optional Applicability information
Definition	Level of formality of the ontology
Domain Range Cardinality	omv:Ontology omv:FormalityLevel 0:1
OMV version	0.9.1
Comments	Pre-defined values. See section B.9

Table B.24: Property: hasFormalityLevel

knownUsage	
Name Type Identifier	knownUsage DatatypeProperty
Occurrence Constraint Category	optional Applicability information
Definition	The applications where the ontology is being used
Domain Range Cardinality	omv:Ontology xsd:string 0:n
OMV version	2.2
Comments	None

Table B.25: Property: knownUsage

hasOntologyLanguage	
Name Type Identifier	hasOntologyLanguage ObjectProperty
Occurrence Constraint Category	required Format information
Definition	The ontology language
Domain Range Cardinality	omv:Ontology omv:OntologyLanguage 1:1
OMV version	0.1
Comments	Pre-defined values. See section B.7

Table B.26: Property: hasOntologyLanguage

hasOntologySyntax	
Name	hasOntologySyntax
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	The presentation syntax for the ontology language
Domain	omv:Ontology
Range	omv:OntologySyntax
Cardinality	0:1
OMV version	0.1
Comments	Pre-defined values. See section B.6

Table B.27: Property: hasOntologySyntax

resourceLocator	
Name Type Identifier	resourceLocator DatatypeProperty
Occurrence Constraint Category	required Availability information
Definition	The location where the ontology can be found. It should be accessible via a URL. It can be the same as the value for URI property
Domain Range Cardinality	omv:Ontology xsd:string 1:n
OMV version	0.1
Comments	None

Table B.28: Property: resourceLocator

version	
Name Type Identifier	version DatatypeProperty
Occurrence Constraint Category	required Availability information
Definition	The version information of the ontology
Domain Range Cardinality	omv:Ontology xsd:string 1:1
OMV version	0.1
Comments	Version information could be useful for tracking, comparing and merging ontologies. It is highly recommended the use of a well defined numbering schema for the version information (e.g. X.Y.Z where X is a major release, Y is minor release and Z is a revision number)

Table B.29: Property: version

hasLicense	
Name	hasLicense
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Availability information
Definition	Underlying license model
Domain	omv:Ontology
Range	omv:LicenseModel
Cardinality	0:1
OMV version	0.1
Comments	Reference to a concrete LicenseModel Pre-defined values. See section B.3

Table B.30: Property: hasLicense

useImports	
Name	useImports
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Relationship information
Definition	References another ontology metadata instance that describes an ontology containing definitions, whose meaning is considered to be part of the meaning of the ontology described by this ontology metadata instance
Domain	omv:Ontology
Range	omv:Ontology
Cardinality	0:n
OMV version	0.1
Comments	Each reference consists of a URI

Table B.31: Property: useImports

hasPriorVersion	
Name	hasPriorVersion
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Relationship information
Definition	Contains a reference to another ontology metadata instance
Domain	omv:Ontology
Range	omv:Ontology
Cardinality	0:1
OMV version	0.1
Comments	The ontology metadata instance which describes an ontology that is a prior version of the ontology described by this ontology metadata instance. It may be used to organize ontologies by versions and is NULL for initial ontology

Table B.32: Property: hasPriorVersion

isBackwardCompatibleWith	
Name Type Identifier	isBackwardCompatibleWith ObjectProperty
Occurrence Constraint Category	optional Relationship information
Definition	The ontology metadata instance which describes an ontology that is a compatible prior version of the ontology described by this ontology metadata instance
Domain Range Cardinality	omv:Ontology omv:Ontology 0:n
OMV version	0.1
Comments	This also indicates that all identifiers from the previous version have the same intended interpretations in the new version

Table B.33: Property: isBackwardCompatibleWith

isIncompatibleWith	
Name Type Identifier	isIncompatibleWith ObjectProperty
Occurrence Constraint Category	optional Relationship information
Definition	The described ontology is a later version of the ontology described by the metadata specified, but is not backward compatible with it. It can be used to explicitly state that ontology cannot upgrade to use the new version without checking whether changes are required.
Domain Range Cardinality	omv:Ontology omv:Ontology 0:n
OMV version	0.1
Comments	None

Table B.34: Property: isIncompatibleWith

numberOfClasses	
Name Type Identifier	numberOfClasses DatatypeProperty
Occurrence Constraint Category	optional Statistic information
Definition	Number of classes in the ontology
Domain Range Cardinality	omv:Ontology xsd:unsignedLong 0:1
OMV version	0.1
Comments	Language specific value

Table B.35: Property: numberOfClasses

numberOfProperties	
Name Type Identifier	numberOfProperties DatatypeProperty
Occurrence Constraint Category	optional Statistic information
Definition	Number of properties in the ontology
Domain Range Cardinality	omv:Ontology xsd:unsignedLong 0:1
OMV version	0.1
Comments	Language specific value

Table B.36: Property: numberOfProperties

numberOfIndividuals	
Name Type Identifier	numberOfIndividuals DatatypeProperty
Occurrence Constraint Category	optional Statistic information
Definition	Number of individuals in the ontology
Domain Range Cardinality	omv:Ontology xsd:unsignedLong 0:1
OMV version	0.1
Comments	Language specific value

Table B.37: Property: numberOfIndividuals

numberOfAxioms	
Name Type Identifier	numberOfAxioms DatatypeProperty
Occurrence Constraint Category	optional Statistic information
Definition	Number of axioms in the ontology
Domain Range Cardinality	omv:Ontology xsd:unsignedLong 0:1
OMV version	0.1
Comments	The meaning of axiom depends on the ontology language. For instance for a RDF(S) ontology it refers to a statement (i.e. triple) and for an OWL ontology it refers to an OWL axiom.

Table B.38: Property: numberOfAxioms

B.2 OntologyType

This class subsumes types of ontologies according to well-known classifications in the Ontology Engineering literature [GPFLC03].

OntologyType	
Name Type Identifier	OntologyType class
Definition	Categorizes ontologies
OMV version	0.3
Comments	None

Table B.39: Class: OntologyType

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required General information
Definition	The name by which an ontology type is formally known
Domain Range Cardinality	omv:OntologyType xsd:string 1:1
OMV version	0.1
Comments	None

Table B.40: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	A short name by which an ontology type is formally known
Domain Range Cardinality	omv:OntologyType xsd:string 0:1
OMV version	0.1
Comments	None

Table B.41: Property: acronym

description	
Name Type Identifier	description DatatypeProperty
Occurrence Constraint Category	optional General information
Definition	Free text description of an ontology type
Domain Range Cardinality	omv:OntologyType xsd:string 0:1
OMV version	0.1
Comments	None

Table B.42: Property: description

B.2.1 Pre-defined ontology types

Individuals of the class `OntologyType` refer to well-known classifications for ontologies in the literature. Currently the OMV model resorts to a classification on the generality levels of the conceptualisation [Gua98, WW02]:

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	General information
Definition	URL for further documentation
Domain	omv:OntologyType
Range	xsd:string
Cardinality	0:1
OMV version	0.2
Comments	None

Table B.43: documentation

definedBy	
Name	definedBy
Type	ObjectProperty, inverseOf(defines)
Identifier	
Occurrence Constraint	optional
Category	General information
Definition	A party that defined the ontology type
Domain	omv:OntologyType
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.44: definedBy

- upper level ontologies describing general, domain-independent concepts e.g. space, time.
- core ontologies describing the most important concepts in a specific domain
- domain ontology describing some domain of the world
- task ontology describing generic types of tasks or activities e.g. selling, selecting.
- application ontology describing some domain in an application-dependent manner

The class can be extended to support additional classifications (e.g. the one in [McG02]).

B.3 LicenseModel

LicenseModel	
Name	LicenseModel
Type	class
Identifier	LM
Definition	A license model describing the usage conditions for an ontology
OMV version	0.3
Comments	None

Table B.45: Class: LicenseModel

name	
Name	name
Type	DatatypeProperty
Identifier	Used Identifier for this entity.
Occurrence Constraint	required
Category	Availability information
Definition	The name by which a license model is formally known
Domain	omv:LicenseModel
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.46: Property: name

acronym	
Name	acronym
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Availability information
Definition	A short name by which a license model is formally known
Domain	omv:LicenseModel
Range	xsd:string
Cardinality	0:1
OMV version	0.1
Comments	None

Table B.47: Property: acronym

description	
Name	description
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Availability information
Definition	Descriptive free text about a license model
Domain	omv:LicenseModel
Range	xsd:string
Cardinality	0:1
OMV version	0.1
Comments	None

Table B.48: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Availability information
Definition	URL for further documentation
Domain	omv:LicenseModel
Range	xsd:string
Cardinality	0:1
OMV version	0.2
Comments	None

Table B.49: documentation

specifiedBy	
Name	specifiedBy
Type	ObjectProperty, inverseOf(specifies)
Identifier	Used Identifier for this element.
Occurrence Constraint	optional
Category	Availability information
Definition	A party that specified the license model
Domain	omv:LicenseModel
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.50: specifiedBy

B.3.1 Pre-defined license models

Individuals of the class `LicenseModel` refer to well-known license models, such as:

- Academic Free License (AFL)
- Common Public License (CPL)
- Lesser General Public License (LGPL)
- Open Software License (OSL)
- General Public License (GPL)
- Modified BSD License (mBSD)
- IBM Public License (IBM PL)
- Apple Public Source License (APSL)
- INTEL Open Source License (INTEL OSL)
- Mozilla Public License (MPL)
- Creative Commons Licenses (CCL)
 - Attribution (by)
 - Attribution-NoDerivs (by-nd)
 - Attribution-NonCommercial-NoDerivs (by-nc-nd)
 - Attribution-NonCommercial (by-nc)
 - Attribution-Noncommercial-Share Alike (by-nc-sa)
 - Attribution-ShareAlike (by-sa)

The class can be extended to support additional classifications.

B.4 OntologyEngineeringMethodology

OntologyEngineeringMethodology	
Name	OntologyEngineeringMethodology
Type	class
Identifier	
Definition	Information about the ontology engineering methodology
OMV version	0.3
Comments	None

Table B.51: Class: OntologyEngineeringMethodology

name	
Name	name
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Other
Definition	The name by which a ontology engineering method is formally known
Domain	omv:OntologyEngineeringMethodology
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.52: Property: name

acronym	
Name	acronym
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Other
Definition	A short name by which a ontology engineering method is known
Domain	omv:OntologyEngineeringMethodology
Range	xsd:string
Cardinality	0:1
OMV version	0.1
Comments	None

Table B.53: Property: acronym

description	
Name	description
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Other
Definition	Free text description of an ontology engineering method
Domain	omv:OntologyEngineeringMethodology
Range	xsd:string
Cardinality	0:1
OMV version	0.6
Comments	None

Table B.54: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Other
Definition	URL for further documentation
Domain	omv:OntologyEngineeringMethodology
Range	xsd:string
Cardinality	0:1
OMV version	0.6
Comments	None

Table B.55: documentation

developedBy	
Name	developedBy
Type	ObjectProperty
Identifier	inverseOf(develops)
Occurrence Constraint	optional
Category	Other
Definition	A party that developed the ontology engineering methodology
Domain	omv:OntologyEngineeringMethodology
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.56: developedBy

B.5 OntologyEngineeringTool

OntologyEngineeringTool	
Name Type Identifier	OntologyEngineeringTool class
Definition	A tool used to create the ontology
OMV version	0.3
Comments	None

Table B.57: Class: OntologyEngineeringTool

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required Other
Definition	The name by which a tool is formally known
Domain Range Cardinality	omv:OntologyEngineeringTool xsd:string 1:1
OMV version	0.1
Comments	None

Table B.58: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional Other
Definition	A short name by which a tool is known
Domain Range Cardinality	omv:OntologyEngineeringTool xsd:string 0:1
OMV version	0.1
Comments	None

Table B.59: Property: acronym

description	
Name Type Identifier	description DatatypeProperty
Occurrence Constraint Category	optional Other
Definition	Free text description of the tool
Domain Range Cardinality	omv:OntologyEngineeringTool xsd:string 0:1
OMV version	0.1
Comments	None

Table B.60: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Other
Definition	URL for further documentation.
Domain	omv:OntologyEngineeringTool
Range	xsd:string
Cardinality	0:n
OMV version	0.2
Comments	None

Table B.61: documentation

developedBy	
Name	developedBy
Type	ObjectProperty
Identifier	inverseOf(develops)
Occurrence Constraint	optional
Category	Other
Definition	The tool developer party
Domain	omv:OntologyEngineeringTool
Range	omv:Party
Cardinality	0:n
OMV version	0.4
Comments	None

Table B.62: developedBy

B.5.1 Pre-defined ontology languages

Individuals of the class `OntologyEngineeringTool` refer to well-known ontology engineering tools, such as:

- Protégé
- SWOOP
- OntoStudio
- Altova SemanticWorks
- OilEd
- IsaViz
- WebODE
- OntoBuilder
- WSMO Studio

B.6 OntologySyntax

OntologySyntax	
Name	OntologySyntax
Type	class
Identifier	
Definition	Information about the syntax used by an ontology
OMV version	0.3
Comments	None

Table B.63: Class: OntologySyntax

name	
Name	name
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Format information
Definition	The name by which an ontology syntax is formally known
Domain	omv:OntologySyntax
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.64: Property: name

acronym	
Name	acronym
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	A short name by which an ontology syntax is known
Domain	omv:OntologySyntax
Range	xsd:string
Cardinality	0:1
OMV version	0.1
Comments	None

Table B.65: Property: acronym

description	
Name	description
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	Free text description of the used syntax
Domain	omv:OntologySyntax
Range	xsd:string
Cardinality	0:1
OMV version	0.6
Comments	None

Table B.66: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	URL for further documentation.
Domain	omv:OntologySyntax
Range	xsd:string
Cardinality	0:1
OMV version	0.6
Comments	None

Table B.67: documentation

developedBy	
Name	developedBy
Type	ObjectProperty
Identifier	inverseOf(develops)
Occurrence Constraint	optional
Category	Format information
Definition	The party who developed the used syntax
Domain	omv:OntologySyntax
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.68: developedBy

B.6.1 Pre-defined ontology syntaxes

Individuals of the class `OntologySyntax` refers to well-known ontology syntax standards, such as:

- OWL/XML
- RDF/XML

The class can be extended to support additional classifications.

B.7 OntologyLanguage

OntologyLanguage	
Name Type Identifier	OntologyLanguage class
Definition	Information about the language in which the ontology is implemented
OMV version	0.3
Comments	None

Table B.69: Class: OntologyLanguage

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required Format information
Definition	The name by which an ontology language is formally known
Domain Range Cardinality	omv:OntologyLanguage xsd:string 1:1
OMV version	0.1
Comments	None

Table B.70: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional Format information
Definition	A short name by which an ontology language is known
Domain Range Cardinality	omv:OntologyLanguage xsd:string 0:1
OMV version	0.1
Comments	None

Table B.71: Property: acronym

description	
Name Type Identifier	description DatatypeProperty
Occurrence Constraint Category	optional Format information
Definition	Free text description of an ontology language.
Domain Range Cardinality	omv:OntologyLanguage xsd:string 0:1
OMV version	0.6
Comments	None

Table B.72: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	URL for further documentation
Domain	omv:OntologyLanguage
Range	xsd:string
Cardinality	0:1
OMV version	0.6
Comments	None

Table B.73: Property: documentation

developedBy	
Name	developedBy
Type	ObjectProperty
Identifier	inverseOf(develops)
Occurrence Constraint	optional
Category	Format information
Definition	The party who developed the language
Domain	omv:OntologyLanguage
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.74: Property: developedBy

supportsRepresentationParadigm	
Name	supportsRepresentationParadigm
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	The representation paradigm supported by the ontology language
Domain	omv:OntologyLanguage
Range	omv:Party
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.75: Property: supportsRepresentationParadigm

hasSyntax	
Name	hasSyntax
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	The syntactical alternatives of the language
Domain	omv:OntologyLanguage
Range	omv:OntologySyntax
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.76: Property: hasSyntax

B.7.1 Pre-defined ontology languages

Individuals of the class `OntologyLanguage` refer to well-known ontology language standards, such as:

- OWL
- OWL-DL
- OWL-Lite
- OWL-Full
- DAML-OIL
- RDF(S)

The class can be extended to support additional classifications.

B.8 KnowledgeRepresentationParadigm

KnowledgeRepresentationParadigm	
Name Type Identifier	KnowledgeRepresentationParadigm class
Definition	Information about a knowledge representation paradigm a particular language adheres to
OMV version	0.9.1
Comments	E. g. Description Logics, Frames

Table B.77: Class: KnowledgeRepresentationParadigm

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required Format information
Definition	The name by which a KR paradigm is formally known
Domain Range Cardinality	omv:KnowledgeRepresentationParadigm xsd:string 1:1
OMV version	0.9.1
Comments	None

Table B.78: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional Format information
Definition	A short name by which a kR paradigm is known
Domain Range Cardinality	omv:KnowledgeRepresentationParadigm xsd:string 0:1
OMV version	0.9.1
Comments	None

Table B.79: Property: acronym

description	
Name Type Identifier	description DatatypeProperty
Occurrence Constraint Category	optional Format information
Definition	Free text description of the knowledge representation paradigm
Domain Range Cardinality	omv:KnowledgeRepresentationParadigm xsd:string 0:1
OMV version	0.9.1
Comments	None

Table B.80: Property: description

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	URL for further documentation
Domain	omv:KnowledgeRepresentationParadigm
Range	xsd:string
Cardinality	0:1
OMV version	0.9.1
Comments	None

Table B.81: documentation

specifiedBy	
Name	specifiedBy
Type	ObjectProperty
Identifier	inverseOf specifies
Occurrence Constraint	optional
Category	Format information
Definition	Author of the KR paradigm
Domain	omv:KnowledgeRepresentationParadigm
Range	omv:Party
Cardinality	0:n
OMV version	0.1
Comments	None

Table B.82: Property: specifiedBy

B.8.1 Pre-defined knowledge representation paradigms

In this version we foresee two main classes of `KnowledgeRepresentationParadigms`:

- Description Logics
- Frames

B.9 FormalityLevel

FormalityLevel	
Name	FormalityLevel
Type	class
Identifier	
Definition	The level of formality of an ontology
OMV version	0.9.1
Comments	According to classifications in the OE literature

Table B.83: Class: FormalityLevel

name	
Name	name
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Applicability information
Definition	The name by which this element is formally known
Domain	omv:FormalityLevel
Range	xsd:string
Cardinality	1:1
OMV version	0.9.1
Comments	None

Table B.84: Property: name

description	
Name	description
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Format information
Definition	Free text description of the formality level
Domain	omv:FormalityLevel
Range	xsd:string
Cardinality	0:1
OMV version	0.9.1
Comments	None

Table B.85: Property: description

B.9.1 Pre-defined formality levels

The pre-defined values for the formality level are based on the work presented in [LM01], which classifies ontologies in a spectrum of definitions according to the detail in their specification as: catalog, glossary, thesauri, taxonomy, frames and properties, value restrictions, disjointness, general logic constraints.

B.10 OntologyTask

OntologyTask	
Name Type Identifier	OntologyTask class
Definition	Information about the task the ontology was intended to be used for
OMV version	0.9.1
Comments	Super-class of classes modelling typical ontology-related tasks

Table B.86: Class: OntologyTask

name	
Name Type Identifier	taskName DatatypeProperty
Occurrence Constraint Category	required Applicability information
Definition	The name by which an ontology task is formally known
Domain Range Cardinality	omv:OntologyTask xsd:string 1:1
OMV version	0.9.1
Comments	None

Table B.87: Property: name

acronym	
Name Type Identifier	acronym DatatypeProperty
Occurrence Constraint Category	optional Applicability information
Definition	A short name by which an ontology task is known
Domain Range Cardinality	omv:OntologyTask xsd:string 0:1
OMV version	0.9.1
Comments	None

Table B.88: Property: acronym

description	
Name Type Identifier	description DatatypeProperty
Occurrence Constraint Category	optional Applicability information
Definition	Free text description of the ontology task
Domain Range Cardinality	omv:OntologyTask xsd:string 0:1
OMV version	0.9.1
Comments	None

Table B.89: Property: description

B.10.1 Pre-defined ontology tasks

Individuals of the class `OntologyTask` refer to particular application scenarios for ontologies, in which the benefits of using ontologies are widely acknowledged. We differentiate among the following tasks:

documentation	
Name	documentation
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Applicability information
Definition	URL for further documentation
Domain	omv:OntologyTask
Range	xsd:string
Cardinality	0:1
OMV version	0.9.1
Comments	None

Table B.90: documentation

AnnotationTask : the ontology is used as a controlled vocabulary to annotate resources and data. This task includes the usage of a semantically rich ontology for representing arbitrarily complex annotation statements on these resources. The task can be performed manually or (semi-)automatically.

ConfigurationTask : the ontology is designed to provide a controlled and unambiguous means to represent valid configuration profiles in application systems. As the aim of the ontology is to support the operationalization of particular system-related processes; this task is performed automatically in that the ontology is processed in an automatic manner by means of reasoners or APIs.

FilteringTask : the task describes at a very general level how ontologies are applied to refine the solution space of a certain problem, such as information retrieval or personalization. The task is targeted at being performed semi-automatically or automatically.

IndexingTask : in this scenario, the goal of the ontology is to provide a clearly defined classification and browsing structure for the information items in a repository. Again, the task can be performed manually by domain experts or as part of an application in an automatic or semi-automatic way.

IntegrationTask : the task characterizes how ontologies provide an integrating environment, an interlingua, for information repositories or software tools. In this scenario the ontology is applied (semi-)automatically to merge between heterogeneous data pools in the same or in adjacent domains.

MatchingTask : the goal of matching is to establish links between semantically similar data items in information repositories. In contrast to the previous task, matching does not include the production of a shared final schema/ontology as a result of aggregating the matched source elements to common elements. W.r.t. the automatization level the range varies from manual to fully-automatical execution.

MediationTask : the ontology is built to reduce the ambiguities between communicating human or machine agents. It can act as a normative model which formally and clearly defines the meaning of the terms employed in agent interactions. In the context of programmed agents, the task is envisioned to be performed automatically.

QueryFormulationTask : the ontology is used in information retrieval settings as a controlled vocabulary for representing user queries. Usually the task is performed automatically in that the concepts of the ontology is are listed in a query formulation front-end in order to allow users to specifies their queries.

QueryRewritingTask : complementary to the query formulation dimension, this task applies ontologies to semantically optimize query expressions by means of the domain knowledge (constraints, subsumption relations etc.) The task can be interpreted as a particular art of filtering information. The task is performed automatically; however, it assumes the availability of patterns describing the transformations at query level.

PersonalizationTask : the ontology is used mainly for providing personalized access to information resources. Individual user preferences w.r.t. particular application settings are formally specified by

means of an ontology, which, in conjunction with appropriate reasoning services, can be directly integrated to a personalization component for filtering purposes. The usage of ontologies in personalization tasks might be carried out in various forms, from a direct involvement of the user who manually specifies ontological concepts which optimally describe his preferences, to the ontological modelling of user profiles.

SearchTask : the task characterizes how ontologies are used to refine common keyword-based search algorithms using domain knowledge in form of subsumption relations. Ontology-driven search is usually performed automatically by means of reasoning services handling particular aspects of an ontology representation language.

B.11 OntologyDomain

OntologyDomain	
Name Type Identifier	OntologyDomain OntologyDomain
Definition	While the domain can refer to any topic ontology it is advised to use one of the established general purpose topic hierarchy like DMOZ or domain specific topic hierarchy like ACM for the computer science domain. Only this way it can ensured that meaningful information about the relation of the domains of two separate ontologies can be deduced
Comments	None

Table B.91: Class: OntologyDomain

URI	
Name Type Identifier	URI DatatypeProperty
Occurrence Constraint Category	required General information
Definition	The URI of the ontology domain
Domain Range Cardinality	omv:OntologyDomain xsd:string 1:1
OMV version	2.1
Comments	None

Table B.92: Property: URI

name	
Name Type Identifier	name DatatypeProperty
Occurrence Constraint Category	required General information
Definition	The name by which an ontology domain is formally known
Domain Range Cardinality	omv:Ontology xsd:string 1:1
OMV version	2.1
Comments	None

Table B.93: Property: name

isSubDomainOf	
Name Type Identifier	isSubDomainOf ObjectProperty
Occurrence Constraint Category	optional Applicability information
Definition	Specifies the domain topic of which this domain topic is a sub domain
Domain Range Cardinality	omv:OntologyDomain omv:OntologyDomain 0:n
OMV version	0.8
Comments	Typically, the domain can refer to established topic hierarchies such as the general purpose topic hierarchy DMOZ or the domain specific topic hierarchy ACM for the computer science domain

Table B.94: Property: isSubDomainOf

B.12 Party

Party	
Name	Party
Type	class
Identifier	
Definition	A party is a person or an organisation
OMV version	0.4
Comments	None

Table B.95: Class: Party

isLocatedAt	
Name	isLocatedAt
Type	ObjectProperty
Identifier	
Occurrence Constraint	optional
Category	Availability Information
Definition	The geographical location of a party
Domain	omv:Party
Range	omv:Location
Cardinality	0:n
OMV version	0.9
Comments	None

Table B.96: Property: isLocatedAt

develops	
Name	develops
Type	ObjectProperty, inverseOf(developedBy)
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	An entity developed by a party
Domain	omv:Party
Range	omv:OntologyEngineeringTool omv:OntologyEngineeringMethodology omv:OntologyLanguage omv:OntologySyntax
Cardinality	0:n
OMV version	2.4
Comments	None

Table B.97: Property: develops

specifies	
Name	specifies
Type	ObjectProperty, inverseOf(specifiedBy)
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	An entity specified by a party
Domain	omv:Party
Range	omv:LicenseModel omv:KnowledgeRepresentationParadigm
Cardinality	0:n
OMV version	2.4
Comments	None

Table B.98: Property: specifies

defines	
Name Type Identifier	defines ObjectProperty, inverseOf(definedBy)
Occurrence Constraint Category	optional Provenance information
Definition	An entity defined by a party
Domain Range Cardinality	omv:Party omv:OntologyType 0:n
OMV version	0.6
Comments	None

Table B.99: Property: defines

endorses	
Name Type Identifier	endorses ObjectProperty, inverseOf(endorsedBy)
Occurrence Constraint Category	optional Provenance information
Definition	An entity endorsed by a party
Domain Range Cardinality	omv:Party omv:Ontology 0:n
OMV version	2.4
Comments	None

Table B.100: Property: endorses

hasAffiliatedParty	
Name Type Identifier	hasAffiliatedParty ObjectProperty
Occurrence Constraint Category	optional Provenance information
Definition	Another party that is affiliated with this party
Domain Range Cardinality	omv:Party omv:Party 0:n
OMV version	0.2
Comments	None

Table B.101: Property: hasAffiliatedParty

createsOntology	
Name Type Identifier	createsOntology ObjectProperty, inverseOf(hasCreator)
Occurrence Constraint Category	optional Provenance information
Definition	An ontology created by a party
Domain Range Cardinality	omv:Party omv:Ontology 0:n
OMV version	0.7
Comments	None

Table B.102: Property: createsOntology

contributesToOntology	
Name	contributesToOntology
Type	ObjectProperty inverseOf (hasContributor)
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	An ontology a party made contributions to
Domain	omv:Party
Range	omv:Ontology
Cardinality	0:n
OMV version	0.7
Comments	None

Table B.103: Property: contributesToOntology

B.13 Person

Person	
Name	Person
Type	class
Identifier	
Definition	A named individual
OMV version	0.1
Comments	Represents an individual responsible for the creation, or contribution to an ontology

Table B.104: Class: Person

lastName	
Name	lastName
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Provenance information
Definition	The surname of a person
Domain	omv:Person
Range	xsd:string
Cardinality	1:1
OMV version	0.2
Comments	None

Table B.105: lastName

firstName	
Name	firstName
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Provenance information
Definition	The first name of a person
Domain	omv:Person
Range	xsd:string
Cardinality	1:n
OMV version	0.2
Comments	None

Table B.106: firstname

eMail	
Name	email
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Provenance information
Definition	The email address of a person
Domain	omv:Person
Range	xsd:string
Cardinality	1:n
OMV version	0.1
Comments	None

Table B.107: eMail

phoneNumber	
Name Type Identifier	phoneNumber DatatypeProperty
Occurrence Constraint Category	optional Provenance information
Definition	The phone number of a person
Domain Range Cardinality	omv:Person xsd:string 0:n
OMV version	0.1
Comments	None

Table B.108: Property: phoneNumber

faxNumber	
Name Type Identifier	faxNumber DatatypeProperty
Occurrence Constraint Category	optional Provenance information
Definition	The fax number of a person
Domain Range Cardinality	omv:Person xsd:string 0:n
OMV version	0.1
Comments	None

Table B.109: faxNumber

isContactPerson	
Name Type Identifier	isContactPerson ObjectProperty, inverseOf(hasContactPerson)
Occurrence Constraint Category	optional Provenance information
Definition	Instance is contact person of an organisation
Domain Range Cardinality	omv:Person omv:Organisation 0:n
OMV version	0.7
Comments	None

Table B.110: isContactPerson

B.14 Organisation

Organisation	
Name	Organisation
Type	class, subclassOf(Party)
Identifier	
Definition	An organisation of some kind
OMV version	0.6
Comments	Represents social institutions such as universities, companies, societies etc.

Table B.111: Class: Organisation

name	
Name	name
Type	DatatypeProperty
Identifier	
Occurrence Constraint	required
Category	Provenance information
Definition	The name by which an organisation is formally known
Domain	omv:Organisation
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.112: Property: name

acronym	
Name	acronym
Type	DatatypeProperty
Identifier	Used Identifier for this entity.
Occurrence Constraint	required
Category	Provenance information
Definition	A short name by which an organisation is known
Domain	omv:Organisation
Range	xsd:string
Cardinality	1:1
OMV version	0.1
Comments	None

Table B.113: Property: acronym

hasContactPerson	
Name	hasContactPerson
Type	ObjectProperty, inverseOf(isContactPerson)
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	A contact person in the organisation
Domain	omv:Organisation
Range	omv:Person
Cardinality	0:n
OMV version	0.6
Comments	None

Table B.114: hasContactPerson

B.15 Location

Location	
Name Type Identifier	Location class
Definition	A location.
OMV version	0.9
Comments	The geographical location of a party. To keep things simple we use only DatatypeProperties instead of introducing classes for country, street, etc.

Table B.115: Class: Location

state	
Name Type Identifier	state DatatypeProperty
Occurrence Constraint Category	optional Provenance information
Definition	The state of a country.
Domain Range Cardinality	omv:Location xsd:string 0:1
OMV version	0.9
Comments	None

Table B.116: Property: state

country	
Name Type Identifier	country DatatypeProperty
Occurrence Constraint Category	optional Provenance information
Definition	The name of the country
Domain Range Cardinality	omv:Location xsd:string 0:1
OMV version	0.9
Comments	Changed the name from land to country

Table B.117: Property: country

city	
Name Type Identifier	city DatatypeProperty
Occurrence Constraint Category	optional Provenance information
Definition	Name of the city (and zip code).
Domain Range Cardinality	omv:Location xsd:string 0:1
OMV version	0.9
Comments	None

Table B.118: Property: city

street	
Name	street
Type	DatatypeProperty
Identifier	
Occurrence Constraint	optional
Category	Provenance information
Definition	Name of the street and number (address).
Domain	omv:Location
Range	xsd:string
Cardinality	0:1
OMV version	0.9
Comments	None

Table B.119: Property: street

Bibliography

- [ABdB⁺05] J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web Rule Language (WRL). <http://www.w3.org/Submission/WRL/>, September 2005. W3C Member Submission.
- [BCH06a] J. Bao, D. Caragea, and V. Honavar. A distributed tableau algorithm for package-based description logics. In *the 2nd International Workshop On Context Representation And Reasoning (CRR 2006), co-located with ECAI 2006*. 2006.
- [BCH06b] J. Bao, D. Caragea, and V. Honavar. Modular Ontologies - A Formal Investigation of Semantics and Expressivity. In R. Mizoguchi, Z. Shi, and F. Giunchiglia, editors, *Asian Semantic Web Conference 2006*, volume 4185 of *LNCS*, pages 616–631. Springer, 2006.
- [BCH06c] J. Bao, D. Caragea, and V. Honavar. On the semantics of linking and importing in modular ontologies. In *I. Cruz et al. (Eds.): ISWC 2006, LNCS 4273 (In Press)*, pages 72–86. 2006.
- [BCH06d] J. Bao, D. Caragea, and V. Honavar. Towards collaborative environments for ontology construction and sharing. In *International Symposium on Collaborative Technologies and Systems (CTS 2006)*, pages 99–108. IEEE Press, 2006.
- [BCM⁺03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York City, New York, 2003.
- [Bec04] D. Beckett. RDF/XML Syntax Specification. <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>, February 2004.
- [BG04] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. <http://www.w3.org/TR/rdf-schema/>, February 2004.
- [BGS⁺03] F. Budinsky, T. J. Grose, D. Steinberg, R. Ellersick, E. Merks, and S. A. Brodsky. *Eclipse Modeling Framework: A Developer's Guide*. Addison Wesley Professional, 2003.
- [BGvH⁺03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *LNCS*, pages 164–179. Springer, October 2003.
- [BKK⁺01] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson. Extending UML to Support Ontology Engineering for the Semantic Web. In M. Gogolla and C. Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *LNCS*, pages 342–260, Toronto, Canada, October 2001. Springer.
- [BKK⁺02] K. Baclawski, M. M. Kokar, P.A. Kogut, L. Hart, J. Smith, J. Letkowski, and P. Emery. Extending the Unified Modeling Language for Ontology Development. *Software and Systems Modeling*, 1(2):142–156, 2002.

- [BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax, 1998.
- [Bro07] S. Brockmans. *Metamodel-based Knowledge Representation*. Phd thesis, University of Karlsruhe (TH), Karlsruhe, Germany, July 2007.
- [BS02] Alexander Borgida and Luciano Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *CoopIS/DOA/ODBASE*, pages 36–53, 2002.
- [BS03] A. Borgida and L. Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics*, 1:153–184, 2003.
- [CDL01] D. Calvanese, G. De Giacomo, and M. Lenzerini. A Framework for Ontology Integration. In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, editors, *Proceedings of the Semantic Web Working Symposium*, pages 303–316, Stanford, CA, July 2001.
- [CDL02] D. Calvanese, G. De Giacomo, and M. Lenzerini. Description Logics for Information Integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2408 of *LNCS*, pages 41–60. Springer, July 2002.
- [CP99] S. Cranefield and M. Purvis. UML as an Ontology Modelling Language. In *Intelligent Information Integration*, volume 23 of *CEUR Workshop Proceedings*, Stockholm, Sweden, July 1999.
- [dHR⁺08] Mathieu d’Aquin, Peter Haase, Sebastian Rudolph, JÓrŽme Euzenat, Antoine Zimmermann, Martin Dzbor, Marta Iglesias, Yves Jacques, Caterina Caracciolo, Carlos Buil Aranda, and Jose Manuel Gomez. D1.1.3 neon formalisms for modularization: Syntax, semantics, algebra. NeOn Deliverable 1.1.3, The Open University, March 2008.
- [dSM06] M. d’Aquin, M. Sabou, and E. Motta. Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In *Workshop on Modular Ontologies*, Athens, Georgia, November 2006.
- [FHK⁺97] J. Frohn, R. Himmeröder, P.-T. Kandzia, G. Lausen, and C. Schleppehorst. FLORID: A Prototype for F-Logic. In W. A. Gray and P. Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering*, page 583, Birmingham UK, April 1997. IEEE Computer Society.
- [Fra03] D. S. Frankel. *Model Driven Architecture*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.
- [GKM05] B. Groszof, M. Kifer, and D. L. Martin. Rules in the Semantic Web Services Language (SWSL): An Overview for Standardization Directions. In *Proceedings of the W3C Workshop on Rule Languages for Interoperability*, Washington, DC, April 2005.
- [GPFLC03] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Springer, 2003.
- [GPS99] Aldo Gangemi, Domenico M. Pisanelli, and Geri Steve. An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.
- [GPS04] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Working with multiple ontologies on the semantic web. In *International Semantic Web Conference*, pages 620–634, 2004.
- [GPS06] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using epsilon-connections. *J. Web Sem.*, 4(1):40–59, 2006.

- [GPSK05] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Automatic partitioning of owl ontologies using -connections. In *Description Logics*, 2005.
- [Gro05] Object Management Group. Unified Modeling Language: Superstructure. <http://www.omg.org/docs/formal/05-07-04.pdf>, July 2005.
- [Gua98] N. Guarino. Formal Ontology and Information Systems. In *Proceedings of the FOIS'98*, pages 3–15, 1998.
- [HEC⁺04] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, and M. Dutra E. Kendall. OWL Full and UML 2.0 Compared. [http://www.itee.uq.edu.au/~sim\\$colomb/Papers/UML-OWLont04.03.01.pdf](http://www.itee.uq.edu.au/~sim$colomb/Papers/UML-OWLont04.03.01.pdf), March 2004.
- [HKS06] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistable SROIQ. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, Lake District, UK, June 2006. AAAI Press.
- [HM05] P. Haase and B. Motik. A Mapping System for the Integration of OWL-DL Ontologies. In *Proceedings of the ACM-Workshop: Interoperability of Heterogeneous Information Systems (IHIS05)*, Bremen, Germany, November 2005.
- [HP05] J. Hartmann and R. Palma. OMV - Ontology Metadata Vocabulary for the Semantic Web, 2005. v. 1.0, available at <http://omv.ontoware.org/>.
- [HPPSH05] I. Horrocks, B. Parsia, P. F. Patel-Schneider, and J. A. Hendler. Semantic Web Architecture: Stack or Two Towers? In F. Fages and S. Soliman, editors, *PPSWR*, volume 3703 of *LNCS*, pages 37–41. Springer, 2005.
- [HPSB⁺04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>, May 2004.
- [HT00] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 399–404. AAAI Press / The MIT Press, 2000.
- [KC04] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. <http://www.w3.org/TR/rdf-concepts/>, February 2004.
- [KF01] M. Klein and D. Fensel. Ontology versioning for the semantic web, 2001.
- [KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the Association for Computing Machinery*, 42(4):741–843, July 1995.
- [KLWZ03] Oliver Kutz, Carsten Lutz, Frank Wolter, and Michael Zakharyashev. E-connections of description logics. In *Description Logics Workshop, CEUR-WS Vol 81*, 2003.
- [KLWZ04] Oliver Kutz, Carsten Lutz, Frank Wolter, and Michael Zakharyashev. E-connections of abstract description systems. *Artif. Intell.*, 156(1):1–73, 2004.
- [Kre98] R. Kremer. Visual Languages for Knowledge Representation. In *Proceedings of 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Banff, Canada, April 1998. Morgan Kaufmann. <http://ksi.cpsc.ucalgary.ca/KAW/KAW98/kremer/>.
- [LM01] O. Lassila and D. McGuinness. The role of frame-based representation on the semantic web. KSL Tech Report Number KSL-01-02, 2001.

- [LTGP04] A. Lozano-Tello and A. Gomez-Perez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2), 2004.
- [McG02] D. L. McGuinness. Ontologies Come of Age. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.
- [MKW04] S. J. Mellor, S. Kendall, and A. Uhl D. Weise. *MDA Distilled*. Addison Wesley Longman Publishing Co., Inc., Redwood City, California, 2004.
- [MPS08a] B. Motik and P. Patel-Schneider. OWL 2 Web Ontology Language: XML Serialization. <http://www.w3.org/TR/2008/WD-owl2-xml-serialization-20081202/>, December 2008.
- [MPS08b] B. Motik and Peter F. Patel-Schneider. OWL 2 Web Ontology Language: Mapping to RDF Graphs. <http://www.w3.org/TR/2008/WD-owl2-mapping-to-rdf-20081202/>, December 2008.
- [MPSG08] B. Motik, P. F. Patel-Schneider, and B. Cuenca Grau. OWL 2 Web Ontology Language: Direct Semantics. <http://www.w3.org/TR/2008/WD-owl2-semantic-20081202/>, December 2008.
- [MPSP08] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax. <http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/>, December 2008.
- [MSS04] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *The Semantic Web - ISWC 2004*, volume 3298 of *LNCS*, pages 549–563, Hiroshima, Japan, November 2004. Springer.
- [Mv03] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Internet : <http://www.w3.org/TR/owl-features/>, August 2003.
- [Obj06] Object Management Group. Meta Object Facility (MOF) Core Specification. <http://www.omg.org/docs/formal/06-01-01.pdf>, January 2006.
- [Ont06] Ontoprise. F-Logic-Programs. http://www.ontoprise.de/documents/tutorial_flogic.pdf, January 2006.
- [Org04] National Information Standards Organization. Understanding metadata. NISO Press, 2004.
- [PBMT05] E. Paslaru Bontas, M. Mochol, and R. Tolksdorf. Case Studies on Ontology Reuse. In *Proceedings of the IKNOW05 International Conference on Knowledge Management*, 2005.
- [PM01] H. S. Pinto and J. P. Martins. A methodology for ontology integration. In *Proc. of the International Conf. on Knowledge Capture K-CAP01*, 2001.
- [PSZ06] J.Z. Pan, L. Serafini, and Y. Zhao. Semantic import: An approach for partial ontology reuse. In *Workshop on Modular Ontologies*, 2006.
- [RVMS99] T. Russ, A. Valente, R. MacGregor, and W. Swartout. Practical experiences in trading off ontology usability and reusability, 1999.
- [SK03] H. Stuckenschmidt and M. C. A. Klein. Integrity and Change in Modular Ontologies. In G. Gottlob, editor, *18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 900–908, Acapulco, Mexico, August 2003.
- [SK04] Heiner Stuckenschmidt and Michel C. A. Klein. Structure-based partitioning of large concept hierarchies. In *International Semantic Web Conference*, pages 289–303, 2004.

- [SP04] Evren Sirin and Bijan Parsia. Pellet: An OWL DL Reasoner. In *Description Logics*, 2004.
- [SR06] J. Seidenberg and A. Rector. Web Ontology Segmentation: Analysis, Classification and Use. In C. Goble and M. Dahlin, editors, *Proceedings of the World Wide Web Conference (WWW)*, Edinburgh, UK, June 2006. ACM Press.
- [SS89] M. Schmidt-Schauss. Subsumption in KL-ONE is Undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning (KR-89)*, pages 421–431, Toronto, Canada, May 1989. Morgan Kaufmann.
- [SSW05a] L. Serafini, H. Stuckenschmidt, and H. Wache. A Formal Investigation of Mapping Languages for Terminological Knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI05*, Edinburgh, UK, August 2005.
- [SSW05b] Heiner Stuckenschmidt, Luciano Serafini, and Holger Wache. Reasoning about ontology mappings. Technical report, Department for Mathematics and Computer Science, University of Mannheim ; TR-2005-011, 2005.
- [ST05] Luciano Serafini and Andrei Taminin. Drago: Distributed reasoning architecture for the semantic web. In *European Semantic Web Conference - ESWC*, pages 361–376, 2005.
- [Stu06] Heiner Stuckenschmidt. Towards multi-viewpoint reasoning with OWL ontologies. In *European Semantic Web Conference*, 2006.
- [SU05] H. Stuckenschmidt and M. Uschold. Representation of Semantic Mappings. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings*, volume 04391, Germany, 2005. IBFI, Schloss Dagstuhl.
- [TBMM04] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition. <http://www.w3.org/TR/xmlschema-1/>, October 2004.
- [TF05] S. Tessaris and E. Franconi. Rules and Queries with Ontologies: a Unifying Logical Framework. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, volume 147 of *CEUR Workshop Proceedings*, Edinburgh, Scotland, UK, July 2005. CEUR-WS.org.
- [TV56] A. Tarski and R. Vaught. Arithmetical Extensions of Relational Systems. *Compositio Mathematica*, 13:81–102, 1956.
- [UHW⁺98] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods. Ontology Reuse and Application. In *Proc. of the Int. Conf. on Formal Ontology and Information Systems FOIS98*, 1998.
- [Ull88] J. D. Ullman. *Principles of Database & Knowledge-Base Systems Volume 1: Classical Database Systems*. W.H. Freeman & Company, 1988.
- [vHPSH01] F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference Description of the DAML+OIL Ontology Markup Language. <http://www.daml.org/2001/03/reference.html>, March 2001.
- [Vol04] R. Volz. *Web Ontology Reasoning with Logic Databases*. Phd thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, <http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2004/wiwi/2>, February 2004.
- [VOM02] R. Volz, D. Oberle, and A. Maedche. Towards a Modularized Semantic Web. In *Semantic Web Workshop*, Hawaii, 2002.

- [W3C05a] *Accepted Papers of the W3C Workshop on Rule Languages for Interoperability*, Washington, DC, April 2005. <http://www.w3.org/2004/12/rules-ws/accepted>.
- [W3C05b] W3C. Rule Interchange Format Working Group Charter. <http://www.w3.org/2005/rules/wg/charter>, 2005.
- [WW02] Y. Wand and R. Weber. Information Systems and Conceptual Modelling: A Research Agenda. *Information Systems Research*, 13(4), 2002.