NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D1.1.1 Networked Ontology Model

Deliverable Co-ordinator: Peter Haase, Sebastian Rudolph, Yimin Wang, Saartje Brockmans

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

Other Authors: Raul Palma (UPM), Jérôme Euzenat (INRIA), Mathieu d'Aquin (OU)

In this deliverable we present 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 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. Additionally, we provide a vocabulary to describe ontology metadata in a network of ontologies, the Ontology Metadata Vocabulary OMV.

Document Identifier:	NEON/2006/D1.1.1/v1.0	Date due:	November 30, 2006
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	November 30, 2006
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

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

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	**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
Universidad Politécnica di Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Richard Benjamins E-mail adress: rbenjamins@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
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	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**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
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Asociación Española de Comercio Electrónico (AECE)** C/lcalde Barnils, Avenida Diagonal 437 08036 Barcelona Spain Contact person: Gloria Tort E-mail address: gtort@fecemd.org
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	**Atos Origin S.A. (ATOS)** Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com

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:

- Universidad Politécnica di Madrid (UPM)
- Institut National de Recherche en Informatique et en Automatique (INRIA)
- Open University (OU)

Change Log

Version	Date	Amended by	Changes
0.1	03-27-2006	Peter Haase	creation
0.2	04-10-2006	Yimin Wang	Requirements section
0.3	04-28-2006	Raul Palma	Ontology Metadata Input
0.5	04-30-2006	Peter Haase	First draft for partners to comment
0.6	06-30-2006	Peter Haase	Incorporating comments and introduction from J. Euzenat
0.7	10-10-2006	Yimin Wang	Incorporating comments from H. Lewen; formatting
0.8	10-15-2006	Peter Haase	Incorporating modularization
0.85	10-28-2006	Peter Haase	Reworking Introduction, modularization
0.9	11-01-2006	Peter Haase	Incorporating comments from J. Euzenat
0.99	11-16-2006	Peter Haase	Incorporating final comments from WP1 partners
1.0	12-15-2006	Peter Haase	Incorporating final comments from internal reviewer

Executive Summary

In this deliverable we present a proposal for the NeOn networked ontology model for representing and managing relations between multiple networked ontologies. As the networked ontology model provides the basis for the entire project, the definition of the model is driven by the requirements from the individual workpackages.

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.

Additionally, we provide a vocabulary to describe ontology metadata in a network of ontologies, the Ontology Metadata Vocabulary OMV. 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

List of Figures

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.

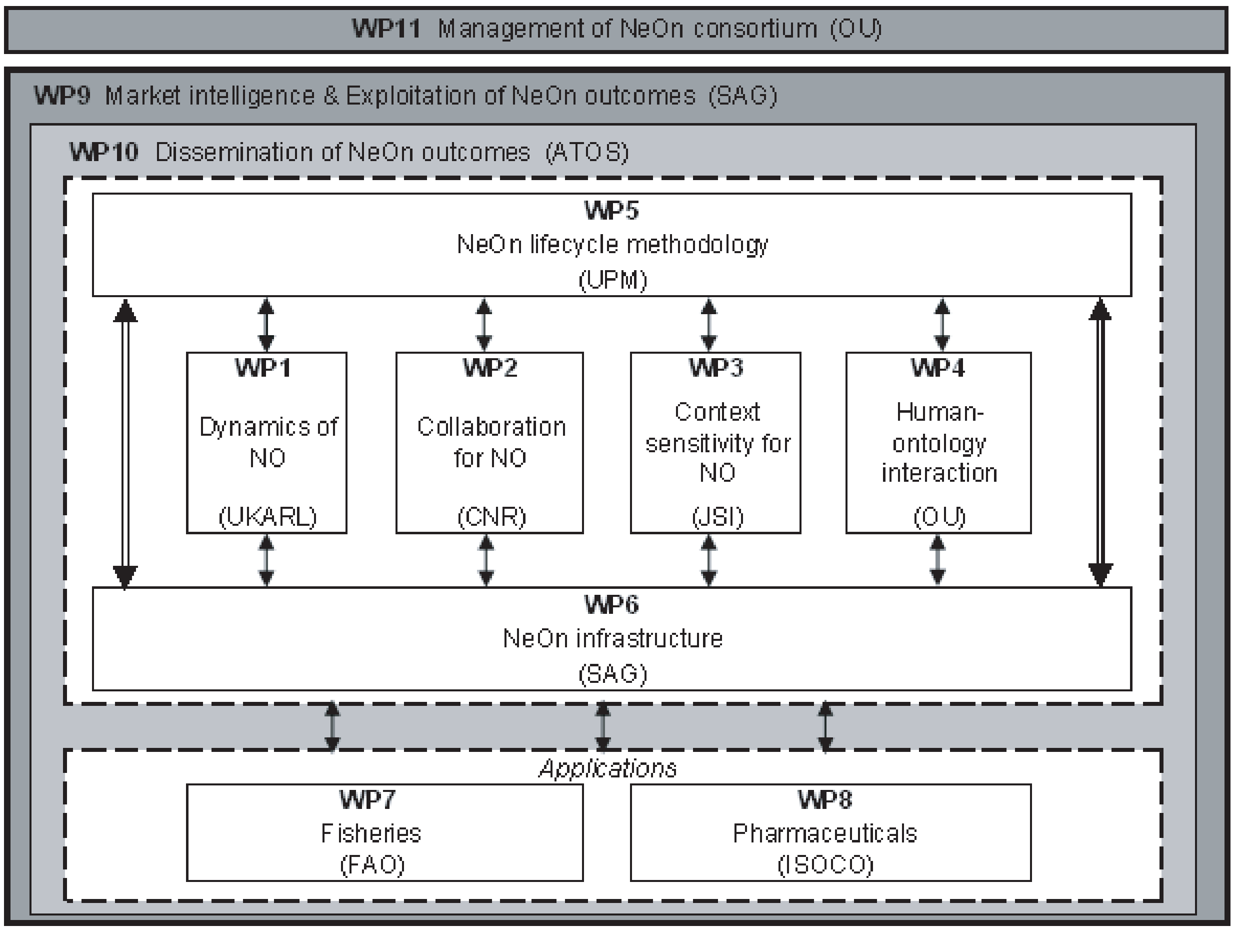
WP11 Management of NeOn consortium (OU)
WP9 Market intelligence & Exploitation of NeOn outcomes (SAG)
WP10 Dissemination of NeOn outcomes (ATOS)
WP5 NeOn lifecycle methodology (UPM)
WP1 Dynamics of NO (UKARL)
WP2 Collaboration for NO (CNR)
WP3 Context sensitivity for NO (JSI)
WP4 Human-ontology interaction (OU)
WP6 NeOn infrastructure (SAG)
Applications
WP7 Fisheries (FAO)
WP8 Pharmaceuticals (ISOCO)

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 on NeOn, current ontology languages lack a number of features to explicitly express the relationships between ontologies and their elements. 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 1.2[1]). 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.

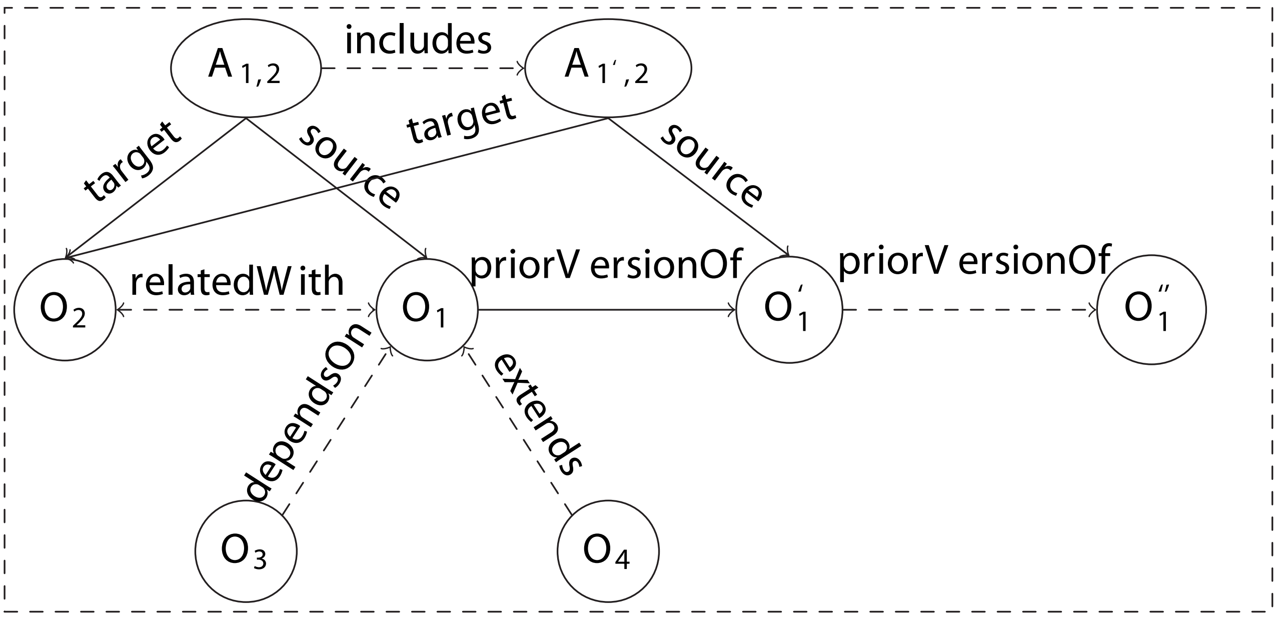

Figure 1.2: The current state of networked ontologies on the web. Dashed relations are in fact not explicitly recorded on the web but can be inferred from user actions.

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.

[1]In 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 it 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 – complementary to 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*.*

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 2.1: 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 2). The term meta-model is chosen, as a meta-model refers

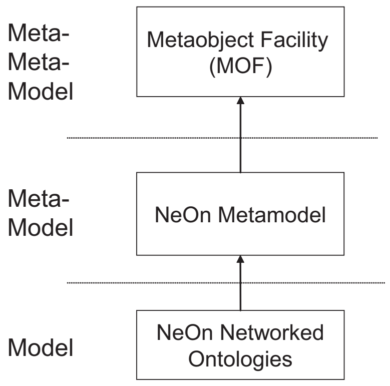

Figure 1.3: The NeOn networked ontology model as a MOF-metamodel

to model of a language, whereas the instances of the meta-model are referred to as models. In our case, models thus refer to the actual ontologies.

The metamodeling features of Model Driven Architecture (MDA, [MKW04]) provide the means for the specification of modeling languages in a standardized, platform independent manner. In short, the Meta Object Facility (MOF, [Obj02]) provides the language for creating metamodels, UML ([Fow03]) 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:

Interoperability with Software Engineering approaches In order for the NeOn reference architecture and toolkit 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 networked ontology model defined in MOF, we can utilize MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels.

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, Java, etc. For example, using the MOF-Java mapping, it is possible to automatically generate a Java API for a MOF metamodel. In NeOn, we use this transformation to generate a Java API for the networked ontology model as part of the NeOn reference architecture developed in WP6.

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 recently has been proposed as a visual notation for knowledge representation languages as well ([HEC+04], [BKK+01], [CP99], [Kre98]). While UML itself lacks specific features of KR languages, the extension mechanisms—UML profiles—allow to tailor the visual notation to specific needs.

Independence from particularities of specific formalisms The metamodeling approach of MDA and MOF allows to define the networked ontology model in an abstract form independently from the particularities of specific logical formalisms. This enables to be compatible with currently competing formalisms (e.g. in the case of mapping languages), for which no standard exists yet. Language mappings, also called groundings, define the relationship with particular formalisms and provide the semantics for the networked ontology model. Furthermore, the extensibility capabilities of MOF allow to add new modules to the metamodel if required in the future.

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.

1.3 Overview of the Deliverable

In Chapter 2, 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

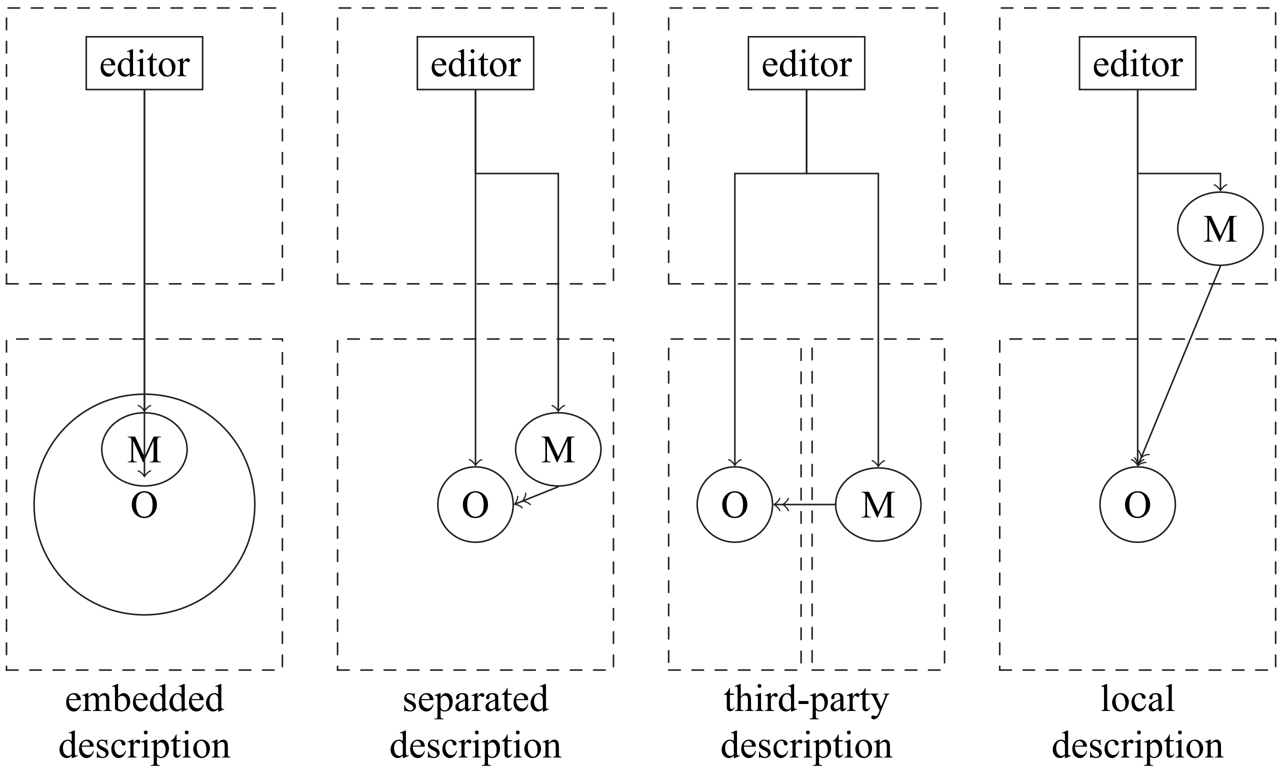

Figure 1.4: The localization of the NeOn networked ontology metadata (M) with regard to the actual ontologies (O). Rectangular boxes refer to machines

the metamodel of OWL in Chapter 3. We then present the metamodel for rule extensions in Chapter 4, for mappings in Chapter 5, and for modularization in Chapter 6.

We then present OMV as a metadata ontology to capture information about networked ontologies in Chapter 7. Finally, in Chapter 8 we provide a summary and an outlook to subsequent deliverables.

1.4 Relationships and Dependencies with other Workpackages

In general, the networked ontology model will be applied throughout the NeOn project, but in each individual workpackage, different aspects of the model are addressed and added. In this section, we discuss the relationships and dependencies of the networked ontology model with other workpackages.

1.4.1 Workpackage 2 – Collaboration

In general, Workpackage 1 and 2 are clearly linked by the general issue of "evolution" of an ontology. This issue involves not only versioning, but also modularization, mapping, and parametrization. The usefulness of possible reasoning tools on these aspects depends on the capacity of the system to express complex versioning patterns, or to describe the structure of the ontology in a rich way, which requires the networked ontology model to include a possibility to represent the different aspects of ontologies.

From the perspective of workpackage 2, following four aspects are mainly related to the work in Workpackage 1.

1. **Networked ontology and its dynamics.** Workpackage 2 essentially needs from Workpackage 1 a notion of networked ontology that is flexible enough to be used in the collaborative and relaxed way that practical modeling of ontologies requires. A collaborative toolkit that supports handling dynamic, contextualized metadata, will provide a listener registration and change notification mechanism for propagating ontology changes into metadata.
2. **KOS reengineering to OWL ontology.** Knowledge Organization Systems (KOS) such as thesauri, terminologies, glossaries, electronic dictionaries and etc. are important sources for ontology construction, but need to be reengineered. The challenge will be: What kind of support could come from WP1 in order to efficiently access and transform KOSes? A possible solution is to create meta-meta-models to map a KOS meta-model to an OWL meta-model or other ontology language formalisms.
3. **Translation support.** A MOF-like approach may be adopted for language interoperability. It would be highly desirable that a close-to-maximal expressivity for the languages to be translated is supported, e.g., by allowing translation of (fragments of) FOL (first order logic) into OWL+Rules. Semantic translation languages might be configured at a higher level, in order to create ad hoc translators between any two logical languages.
4. **Reasoning support.** For successful collaboration between ontology designers, a very lightweight reasoning service should be provided, even at development time, and even at the cost of giving up some completeness.

1.4.2 Workpackage 3 – Context

The networked ontology model is closely related to the notion of context developed as part of Workpackage 3.

1. The networked ontology model needs to be able to represent context as defined in the workpackage, which means, the context information should be represented as a language formalism that is contained in the networked ontology model.
2. The network of ontologies in which a particular ontology is embedded, can itself be a special form of context: For an ontology K (or knowledge base), we assume that there are other ontologies which are related to K in various networked ways, including versioning and mapping information, etc. Then other ontologies together with these links can be understood as a context for K, as they will (in some cases) alter the knowledge which can be inferred from K.

3. The context information will also be used in mapping ontological information, and possibly help in defining the different ontology modules based on different contexts.

1.4.3 Workpackage 4 – Human Ontology Interaction

The objective of Workpackage 4 is to find a proper way to build fine-grained ontologies. The networked ontology model calls for a new generation of human ontology interaction.

In the networked ontological scenario, humans are not only required to perform traditional tasks to collect concepts and construct axioms, etc., but they are also asked for the definition of mapping, modularization and parameters of an ontology. These notions are provided by the networked ontology model proposed in this deliverable.

Otherwise, users should also have a good understanding of the networked ontology model, e.g. of its structure, its meaning. Therefore, the challenge in designing the networked ontology model is to make it understandable intuitive to the users.

1.4.4 Workpackage 5 – Methodology

First, WP5 will use the definition of networked ontologies (provided by WP1) to identify the activities to be included in the networked development process and in the ontology lifecycle. Besides, WP5 will use the work related to dynamics of networked ontologies in the methodology for building networked ontologies.

Although there are many ways to build ontologies, there is still no very efficient and effective methodology to build ontologies in a networked environment with strong emphasis on the dynamic aspects and distribution. This is one of the major requirement of NeOn and also should be carefully considered in WP1 while modeling the networked ontologies. WP5 is going to provide such a methodology to support the lifecycle of networked ontologies.

1.4.5 Workpackage 6 – Infrastructure

The NeOn infrastructure relies on the scientific and technical outcome from WP1, WP2, WP3 and WP4. WP1 will contribute the overall model and the corresponding partial reasoning support to the infrastructure work package. To be specific:

1. The applications developed in NeOn will build upon NeOn infrastructure and aim to support the networked ontology lifecycle development. Thus, the NeOn infrastructure is depending on the networked ontology model as input from Workpackage 1 in order to have good support to handle networked ontologies.

2. The development of components relying on the networked ontology model need to be aligned with the architecture developed inWorkpackage 6.

1.4.6 Workpackage 7 – Fishery Case Study

In WP7, partners will design and implement ontologies in the fisheries domain that can become widely-used standards and references for the fishery communities around the world.

To increase information accessibility to and from non-FAO resources, the ontology mapping and modularization techniques will be helpful to identify and handle the relations between ontologies with different natural languages.

The mappings between multilingual ontologies are important for the WP7 fishery case study. One scenario is: One ontology has mappings to another ontology in a different language, from which it is translated. The relationships between these two ontologies need to be formally defined by the networked ontology model. Moreover, if these two ontologies are connected for some specific needs (e.g., applications requiring multilingual information), it is important to have an appropriate use of the ontologies based on their languages to make sure the knowledge captured is equal in both languages (e.g. Concept A_E in English should be equal to concept A_S in Spanish if A_E is a correct translation of A_S within the context of this ontology.) This scenario has not yet occurred within FAO, but could happen in the future.

Modularizing the fishery ontology with respect to regional or economic regional differences is probably useful. Further, a modularization according to natural languages might be promising. For example, having one multilingual ontology, where concepts have names (attributes) in several languages (multilingual layers). The structure of the ontology is meant to be independent of the multilingual layer (which means it has been modularized), possibly with a strict 1-1 correspondence between the various languages and single ownership of the entire ontology.

A particular use case is the AGROVOC thesaurus (which is to be converted into an ontology). AGROVOC is built by agronomy experts in a collaborative setting, and does not grow simultaneously in all languages. Given its collaborative nature, it needs a modularization tool (in terms both of structure and language layers, to be able to have different authors and/or institutions work on it) that is built on the modularization model.

1.4.7 Workpackage 8 – Pharmaceutical Case Study

Invoicing Management

The main goal of the invoicing management use case is to enable pharmacies in Spain to exchange invoices with wholesalers electronically without having to adapt to the wholesalers' invoice exchange standard first. This is especially challenging since all recent approaches to unify invoice exchange at a larger scale have failed. To support the vision of real time invoice exchange between parties not sharing a common standard, several technologies have to be provided by WP1:

1. **The Networked Ontology Model** supplies the formal grounding of the technologies applied in the case study. It will define the ontology language, the representation and interpretation of rules, ontology mappings and modules as well as the corresponding operators and algebra for modular ontologies. For annotation purposes, an ontology metadata standard will also be developed. Using these techniques, ontologies (as representations of the local invoice data schema of pharmacies or wholesalers) can be mapped and matched and even background knowledge about past transactions can be exploited. For the pharmacies it will thus become easier to exchange invoices. Ideally, the complexity of the underlying problem is hidden completely so to the users exchanging invoices is simply a click.

2. **Dynamics and Change Propagation** are important because it can be expected that external factors are subject to change. Legislation could be changed for instance, or an important new feature could become mandatory for electronic invoices. WP1 will provide methods for ontology evolution, change propagation and ontology versioning. It will provide the features required to keep ontologies across partners consistent and synchronized by keeping track of different versions and propagating changes.

3. **Consistency Models:** Due to the fact that we have to deal with multiple ontologies instead of just a single global one, it can be expected that inconsistencies will occur and will have to be resolved. WP1 will thus address consistency models and inconsistency handling, both for single as well as multiple ontologies. That way, the reasoning needed for adaptive behavior can be performed.

4. **Management of Networked Ontologies:** This refers to the framework holding the different technologies together and providing functionalities like reasoning over networked ontologies and management of ontology metadata. Without the ability to reason over the boundary of single ontology, using mappings and modules does not make sense. So it is of great importance to enable distributed reasoning. Also, ontology metadata has to be generated and stored.

5. **Dynamics of Metadata:** Since most of the ontologies are not static but will likely evolve and be changed over time, it is important to discover and propagate changes.

Semantic Nomenclator

The goal of this case study is to facilitate the work of the General Spanish Council of Pharmacists (GSCoP) which has to maintain the Health Information Data Base (BOT Plus) for health professionals. This database comprises diverse information about medicines and patients. It provides access to information on diseases, symptoms, treatments and so on. The entries are harmonized, so access is possible within any pharmacy in Spain. The problem is that there are constant updates of the information in the database but no systematized approach for creating, maintaining and updating information. Data is provided by the pharmacies in diverse formats and protocols, leading to manual updates which take long time and are error-prone. Using NeOn technology, these obstacles can be overcome by automatizing knowledge processing and consistency checks. The technologies provided by WP1 are the same as for the invoice management scenario, since the idea is to rely on a common infrastructure for both use cases. The emphasis however lies more on versioning, evolution, data integration, and consistency handling.

In the domains of both of the above case studies, the application development has previously been driven by technologies from software engineering, rather than ontology engineering. The use of model driven approaches has been underrated thus far. The technologies of Model Driven Architectures and a networked ontology model defined in terms of the standards of MDA is expected provide the interoperability with current software engineering approaches and to enable an easier transition from current technologies to semantics-based techniques.

Part II

The NeOn Metamodel for Networked Ontologies

Chapter 2

Metamodeling for Ontologies

2.1 Metamodeling with MOF

This section introduces the essential ideas of MOF and shows how a metamodel and a UML profile for networked ontologies fit into this more general picture. The need for a dedicated visual ontology modeling language stems from the observation that an ontology cannot be sufficiently represented in UML [HEC^{+}04]. The two representation mechanisms share a set of core functionalities such as the ability to define classes, class relationships, and relationship cardinalities. But despite this overlap, there are many features which can only be expressed in OWL, and others which can only be expressed in UML. Examples for these differences in expressiveness are transitive and symmetric properties in OWL or methods in UML. For a full account of the conceptual differences we refer the reader to [IBM05].

UML methodology, tools and technology, however, seem to be a feasible approach for supporting the development and maintenance of networked ontologies.

The general idea of using MOF-based metamodels and UML profiles for this purpose is depicted in Figure 2.1: A metamodel for networked ontologies as well as a UML profile are grounded in MOF, in that they are defined in terms of the MOF meta-metamodel, explained further in this section. The UML profile mechanism is an extension mechanism to tailor UML to specific application areas. Our proposed UML profile defines a visual notation for optimally supporting the specification of ontologies, rules and ontology mappings. This visual syntax is based on the metamodel and is independent from a concrete mapping formalism. Mappings in both directions between the metamodel and the profile have to be established.

OWL DL ontologies, rules, and ontology mappings in a concrete language instantiate the metamodel. The constructs of the specific languages have a direct correspondence with those of the metamodel. Analogously, specific UML models instantiate the UML profile. Within the MOF framework, the UML models are translated into definitions based on the above mappings between the metamodel and the UML profile. In case of ontology mappings, the decision about a concrete mapping formalism is taken in this translation step, so after the visual modeling of the ontology mappings. This decision is based on the types of the mappings which were modeled.

2.1.1 Meta Object Facility

The Meta Object Facility (MOF) is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. The goal is to provide a framework that supports any kind of metadata and that allows new kinds to be added as required. MOF plays a crucial role in the four-layer metadata architecture of the Object Management Group (OMG) shown in Figure 2.2. The bottom layer of this architecture encompasses the raw information to be described. For example, Figure 2.2 contains information about a wine called ElyseZinfandel and about the Napa region, where this wine grows. The model layer contains the definition of the required structures, e.g. in the example it contains the classes used for grouping information. Consequently, the classes Wine and Region are defined. If these

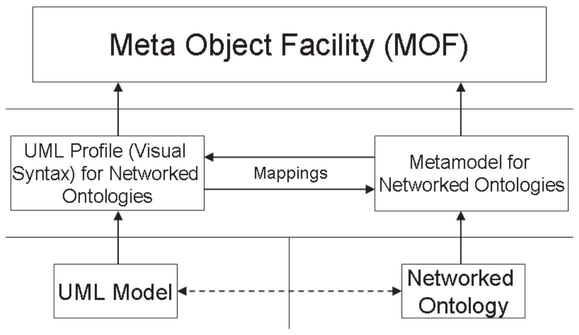
Meta Object Facility (MOF)
UML Profile (Visual Syntax) for Networked Ontologies
Mappings
Metamodel for Networked Ontologies
UML Model
Networked Ontology

Figure 2.1: How a metamodel and a UML profile for ontologies fit into the picture of the Meta Object Facility framework

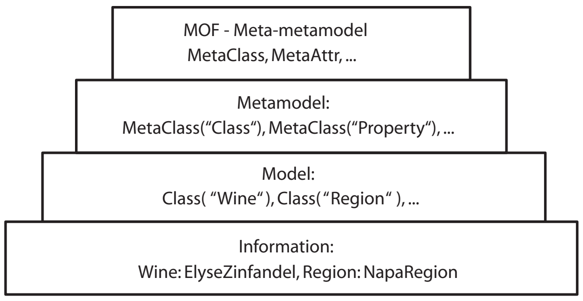
MOF - Meta-metamodel
MetaClass, MetaAttr, ...
Metamodel:
MetaClass("Class"), MetaClass("Property"), ...
Model:
Class("Wine"), Class("Region"), ...
Information:
Wine: ElyseZinfandel, Region: NapaRegion

Figure 2.2: OMG Four Layer Metadata Architecture

are combined, they describe the model for the given domain. The metamodel 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 meta classes. Finally, the MOF constitutes the top layer, also called the meta-metamodel layer. 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 UML metamodel or the metamodel for OWL DL ontologies, SWRL rules and ontology mappings.

2.1.2 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.

2.1.3 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

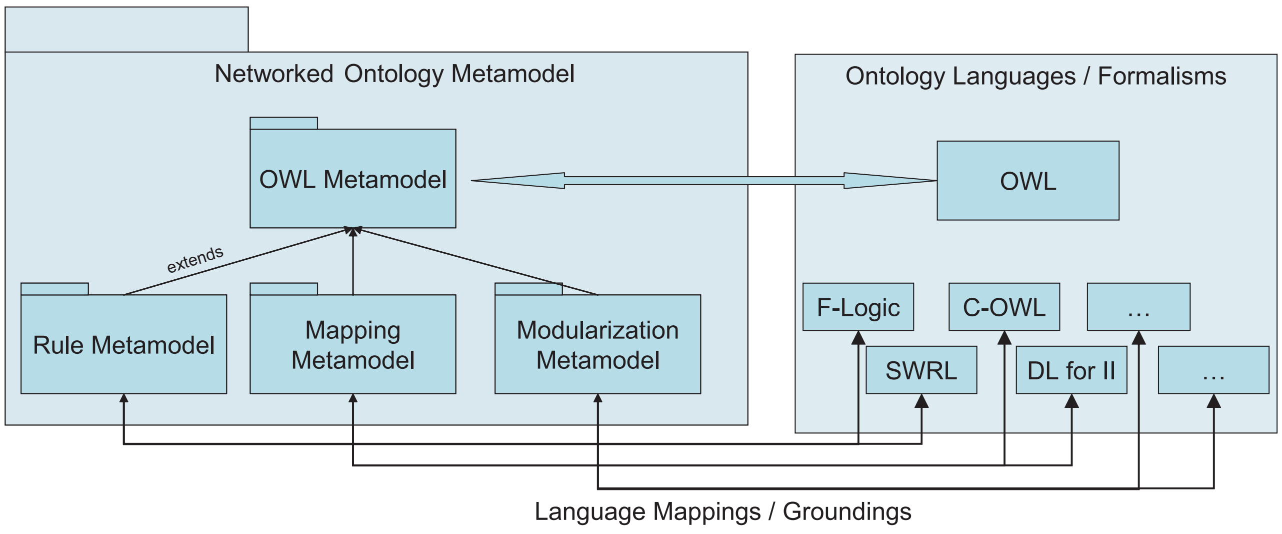

Figure 2.3: Modules of the Networked Ontology Metamodel and possible groundings in ontology languages

of modeling constructs. Thus, MOF metamodels look like UML class models, and one can use UML class-modeling tools to create them.

2.2 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 2.3.

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 subsequent deliverables (D1.1.2 and D1.1.3) we will define the appropriate semantics for modular ontologies in NeOn.

Chapter 3

The OWL Metamodel

3.1 Design considerations

A metamodel for an ontology language can be derived from the modeling primitives offered by that language. The formal semantics of OWL is derived from Description Logics (DL), an extensively researched KR formalism. Hence, most primitives offered by OWL can also be found in a Description Logic. Three species of OWL have been defined. One variant called OWL Full can represent arbitrary RDF components inside of OWL documents. This allows, for example, to combine the OWL language with arbitrary other representation languages. From a conceptual perspective a metamodel for OWL Full necessarily has to include elements for the representation of RDF.

Another variant called OWL DL states syntactic conditions on OWL documents, which ensure that only the primitives defined within the OWL language itself can be used. OWL DL closely corresponds to the SHOIN(D) description logic and all language features can be reduced[1] to the primitives of the SHOIN(D) logic. Naturally, a metamodel for OWL DL is smaller and less complex than a metamodel for OWL Full. Similarly, an OWL DL metamodel can be built in a way such that all elements can be easily understood by people familiar with description logics. A third variant called OWL Lite disallows some constructors of OWL DL, 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][Corollary 3.4.1]. Hence, a metamodel for OWL DL necessarily includes OWL Lite.

While OWL Full is the most expressive of the members of the OWL family that has mainly been defined for compatibility with existing standards such as RDF, OWL Full is undecidable and thus impractical for applications that require complete reasoning procedures. OWL DL as a sublanguage was designed to regain computational efficiency, which is deemed important in the NeOn project. For OWL DL practical reasoning algorithms are known, and increasingly more tools support this or slightly less expressive languages. We therefore decided for OWL DL as the core of the metamodel for NeOn networked ontologies.

3.2 A Metamodel for OWL DL

This section will provide an overview of the OWL language whilst introducing our metamodel. Interested readers may refer to the specifications [Mv03] for a full account of OWL. 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. It primarily uses basic well-known concepts from UML2. Additionally, the metamodel is augmented with constraints, expressed in the Object Constraint Language ([WK04]), specifying invariants that have to be fulfilled by all models that instantiate the metamodel. Constraints are given in footnotes.

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

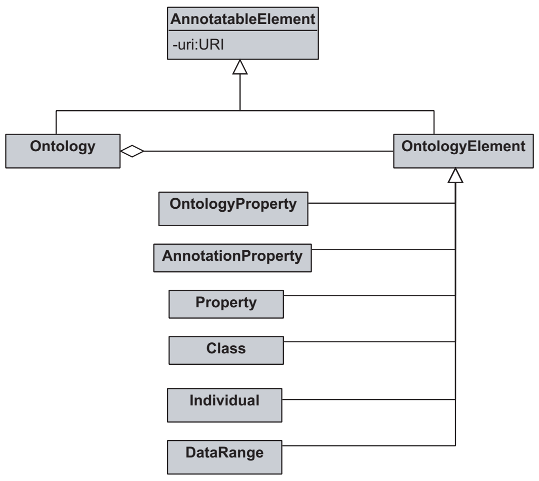

Figure 3.1: Main Elements of the OWL DL Metamodel

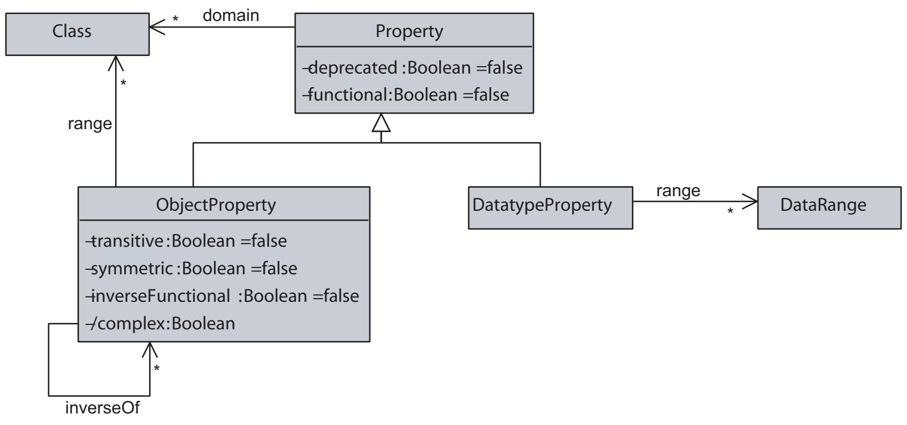
Class
domain
Property
-deprecated :Boolean =false
-functional:Boolean =false
range
ObjectProperty
-transitive:Boolean =false
-symmetric:Boolean =false
-inverseFunctional :Boolean =false
-/complex:Boolean
inverseOf
DatatypeProperty
range
DataRange

Figure 3.2: Properties

Ontologies

URIs are used to identify all objects in OWL. Figure 3.1 shows the central part of the OWL DL metamodel. Among others, it shows that every element of an ontology is a subclass of the class `OntologyElement` and hence a member of an `Ontology`.

Properties

Properties represent named binary associations in the modeled knowledge domain. OWL distinguishes two kinds of properties, so called object properties[2] and datatype properties[3]. Figure 3.2 shows that both

[2]Object properties can only be subproperties of other object properties: `subPropertyOf->forAll(oclIsTypeOf(ObjectProperty))`.

[3]Datatype properties can only be subproperties of other datatype properties: `subPropertyOf->forAll(oclIsTypeOf(DatatypeProperty))`.

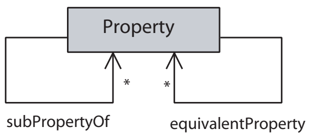

Figure 3.3: Property axioms

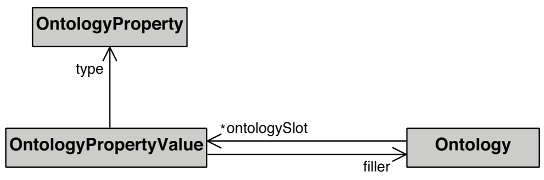

Figure 3.4: Property axioms

are generalized by the abstract metaclass `Property`. Properties can be functional, i.e. their range may contain at most one element. Their domain is always a class. Object properties may additionally be inverse functional, transitive, symmetric or inverse to another property. Their range is `Class`[4][5], while the range of datatype properties is `Datarange`.

Users can relate properties by using two axioms, modeled as in Figure 3.3. Property subsumption (`subPropertyOf`)[6] specifies that the extension of a property is a subset of the related property. Similarly, property equivalence (`equivalentProperty`)[7] defines extensional equivalence. OWL DL disallows that object and datatype properties are related via axioms.

Ontology properties

Ontologies themselves can have properties, which are represented via the `OntologyProperty` metaclass. For example, the ontology property `owl:imports` allows to logically include the elements of one ontology in another ontology. OWL DL predefines several ontology properties and allows users to define further ontology properties. A concrete instance of an ontology property is represented through `OntologyPropertyValue`, which instantiates a certain type of `OntologyProperty` and is a reference between two ontologies (cf. Figure 3.4).

Annotation properties

Given elements of an OWL ontology can be annotated with metadata. Several annotation properties, e.g. `owl:versionInfo`, are predefined and users can define further annotation properties. We treat annotation properties similarly to ontology properties. However, the subject of an `AnnotationPropertyValue` is an `AnnotateableElement` and the object is a Annotation, which can be either a `DataValue`, a `URI` or an `Individual` (cf. Figure 3.5).

Class Constructors

In comparison to UML, OWL DL does not only allow to define simple named classes. Instead, classes can be formed with several class constructors (cf. Figure 3.6). One can conceptually distinguish the boolean combination of classes, restrictions and enumerated classes. `EnumeratedClass` is only available in OWL

[4]A property is complex, when it is functional or inverse functional, when it is used in a (unqualified) cardinality restriction, when it contains a complex inverse, or when it has a complex superproperty:

```
complex=functional or inverseFunctional
or NumberRestriction.allInstances()->exists(onProperty=self)
or inverseOf->exists(complex)
or subPropertyOf->exists(complex).
```

[5]OWL DL mandates that no complex property may be transitive:

```
complex implies not transitive.
```

[6]This association is transitive:

```
Property.allInstances()-> forAll(r,s,t|(r.subPropertyOf->includes(s)
and s.subPropertyOf->includes(t) implies r.subPropertyOf->includes(t))).
```

[7]Every equivalent property is a superproperty:

```
subPropertyOf->includesAll(equivalentProperty).
```

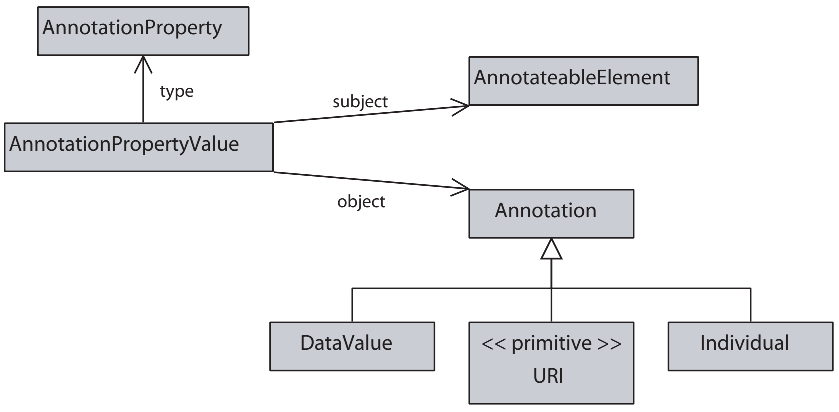

Figure 3.5: Annotations

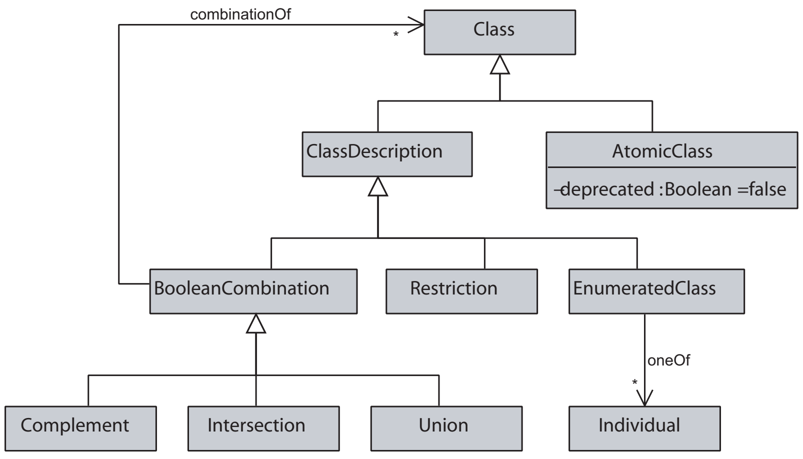

Figure 3.6: Class constructors

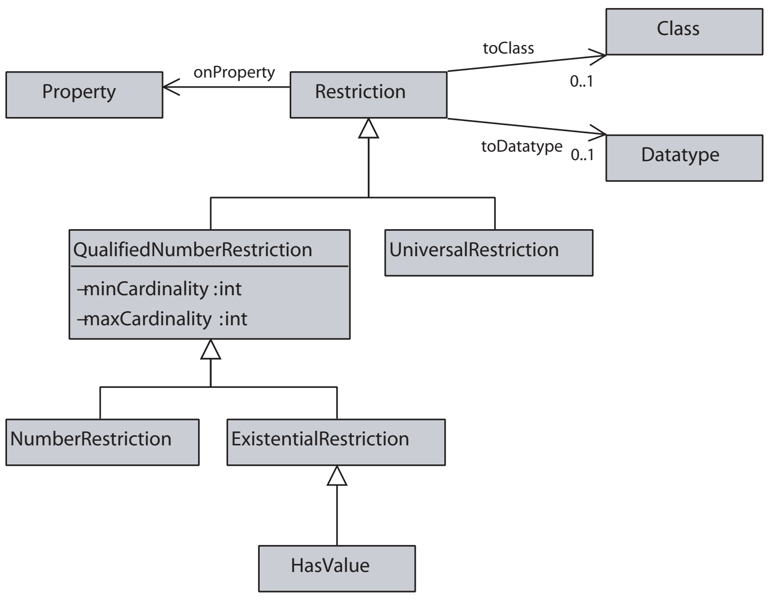

Figure 3.7: OWL Restrictions

DL and is defined through a direct enumeration of named[8] individuals. Boolean combinations of classes are provided through `Complement`[9], `Intersection` and `Union`.

Restrictions are class constructors that restrict the range of a property for the context of the class (cf. Figure 3.7). Restrictions can be stated *on* datatype and object properties. Accordingly they limit the value *to* a certain datatype or class extension[10][11][12]. `UniversalRestriction` provides a form of universal quantification that restricts the range of a class to the extension of a certain class or datatype[13]. We introduce an abstract metaclass `QualifiedNumberRestriction` to relate unqualified cardinality restrictions (which are available in OWL) and existential restrictions. Obviously the minimum cardinality is by default 0 and may not be negative[14] while the maximum cardinality should not be smaller than the minimum cardinality[15], even though OWL allows this by making the class definition become inconsistent. Unqualified number restrictions (`NumberRestriction`) are available in OWL and define how many elements the range of the given property has to have while not restricting the type of the range[16]. `ExistentialRestricion` can logically and semantically be seen as a special type of qualified number restrictions where the cardinality is fixed[17]. OWL also provides `HasValue`, which is a special type of existential restriction allowing us to specify classes based on the existence of particular property values[18].

[8]The enumerated individuals are not anonymous:
`oneOf->forAll(name.notEmpty()).`

[9]Exactly one class is involved:
`combinationOf->size()=1.`

[10]The value can be either one class or one datatype:
`toClass->size()=1 xor toDatatype->size()=1.`

[11]The value of a datatype property can only be a datatype:
`onProperty.ocllsKindOf(DatatypeProperty) implies toDatatype->size()=1.`

[12]The value of an object property can only be a class:
`onProperty.ocllsKindOf(ObjectProperty) implies toClass->size()=1.`

[13]The reader may note that this is logically not understood as a constraint but as an entailment rule.

[14]The minimum cardinality is nonnegative:
`minCardinality>=0.`

[15]The minimum cardinality should not excess the maximum cardinality:
`maxCardinality>=minCardinality.`

[16]The cardinality restriction is unqualified:
`toClass=owl::Thing` or `toDatatype=rdfs::Literal.`

[17]The minimum cardinality is 1, the maximum cardinality is infinite:
`minCardinality=1 and maxCardinality=*.`

[18]
```
toClass.oclssTypeOf(EnumeratedClass)
and toClass.oclAsType( EnumeratedClass).oneOf->size()=1)
or (toDatatype.ocllsTypeOf(EnumeratedDatatype)
```

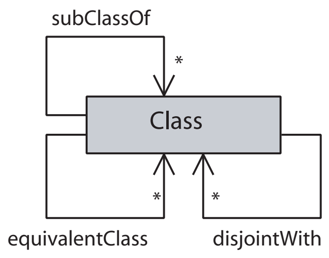

Figure 3.8: Class axioms

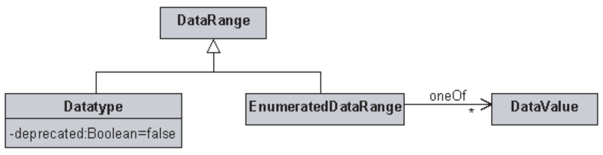

Figure 3.9: Data types

Figure 3.8 shows that classes can be related with each other using class axioms, such as class subsumption (`subClassOf`)[19][20][21], class equivalence (`equivalentClass`)[22] and class disjointness (`disjointWith`). These relations between classes are naturally modelled as associations.

Datatypes

The datatype system of OWL is provided by XML Schema, which provides a predefined set of named datatypes (`PrimitiveType`), e.g. strings `xsd:string`. Additionally users may specify a range of data values (`EnumeratedDataRange`), using the `oneOf` construct (cf. Figure 3.9).

Knowledge Base

OWL does not follow the clear conceptual separation between terminology (T-Box) and knowledge base (A-box) that is present in most description logics and in MOF, which distinguishes between model and information. The knowledge base elements (cf. Figure 3.10) are part of an ontology. An `Individual` is an instantiation of a `Class` and is the subject of a `PropertyValue`, which instantiates a `Property`. Naturally, an `ObjectPropertyValue`[23] relates its subject with another `Individual` whilst a `DatatypePropertyValue`[24] relates its subject with a `DataValue`, which is an instance of a primitive datatype.

Individuals[25] can be related via three axioms, as shown in Figure 3.11. The `sameAs`[26] association allows

`and toDatatype.oclsAsType(EnumeratedDatatype).oneOf->size()=1).`

[19]The subclass relation is transitive:
`Class.allInstances()->forAll(b,c,d|(b.subClassOf->includes(c) and c.subClassOf->includes(d)) implies b.subClassOf->includes(d)).`

[20]Every class is a subclass of owl:Thing:
`subClassOf->includes(owl::Thing).`

[21]owl:Nothing is a superclass of every class:
`owl::Nothing.subClassOf->includes(self).`

[22]Every equivalent class is trivially a superclass:
`subClassOf->includesAll(equivalentClass).`

[23]This can only be an instance of an object property:
`type.oclIsTypeOf(ObjectProperty).`

[24]This can only be an instance of a datatype property:
`type.oclIsTypeOf(DatatypeProperty).`

[25]Every Individual is an instance of owl:Thing:
`type->includes(owl::Thing).`

[26]This attribute is symmetric:
`sameAs->forAll(sameAs(self)).`

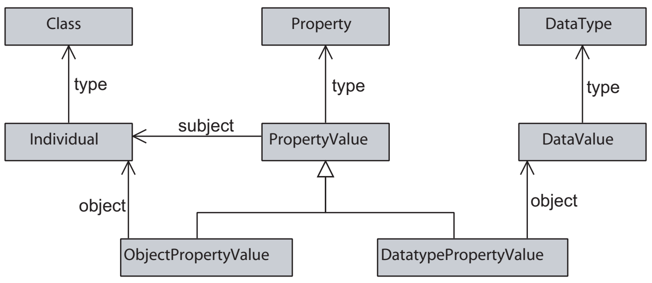

Figure 3.10: Knowledge Base Items

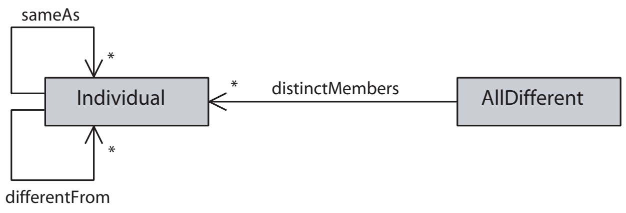

Figure 3.11: Knowledge Base Axioms

users to state that two individuals (with different names) are equivalent. The `differentFrom`[27] association specifies that two individuals are not the same[28]. `AllDifferent` is a simpler notation for the pairwise difference of several individuals.

[27]This feature is symmetric:
`differentFrom->forAll(differentFrom(self)).`
[28]The reader may note that OWL does not take the unique names assumption.

Chapter 4

The Rule Metamodel

4.1 Design Considerations

Although OWL DL is very expressive, it is a *decidable* fragment of first-order logic, and thus cannot 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 do not share this restriction, but lack some of the expressive power of OWL DL: 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]. Just recently the W3C has chartered a working group for the definition of a Rule Interchange Format [W3C05b]. One of the most prominent proposals for an extension of OWL DL 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. We extend our metamodel for OWL DL presented in the previous chapter with a metamodel for SWRL that directly resembles the extensions of SWRL to OWL DL.

While we consider SWRL as the most relevant rule extension for OWL in the context of NeOn, other rule languages may become relevant as well. In the following we briefly discuss how such languages could at a later stage be incorporated in our approach.

4.1.1 Possible Directions to Cover other Rule Languages

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 by our metamodel. Using additional constraints (e.g. in OCL) it can be checked whether a rule is DL-safe. It should be noted that SWRL is not the only rule language which has been proposed for ontologies. Other prominent alternatives for rule languages are mentioned in the W3C Rule Interchange Format Working Group charter [W3C05b], 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 define different metamodels, each of which is tailored to a specific rules language.

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 the SWRL metamodel defined here by augmenting it with some features to allow for the modeling of language primitives which are not present in SWRL. As a result, one would end up with a common metamodel for different rules languages. An advantage of the latter approach would be a gain in flexibility. Overcoming the intricate semantic differences between different ontology and rule languages – that are often difficult to understand for the practitioner – is an ongoing research effort. Building on recent advances in this field, e.g. [MHRS06] and we are confident of extending the networked ontology model to integrate both logic programming based languages (F-Logic) and DL-based languages (OWL).

4.2 A Metamodel for SWRL Rules

We propose a metamodel for SWRL rules as a consistent extension of the metamodel for OWL DL ontologies which we described in the previous section of this paper. Figure 4.1 shows the metamodel for SWRL rules. We discuss the metamodel step by step along the SWRL specifications. To state the rule metamodel more precisely, we augment it with appropriate OCL constraints. We list them in footnotes and explain their semantics. Interested readers may refer to the specifications [HPSB^{+}04] for a full account of SWRL. For a complete reference of the formal correspondence between the metamodel and SWRL itself, we refer the reader to [BH06b].

4.2.1 Rules

SWRL defines rules as part of an ontology. The SWRL metamodel – as shown in Figure 4.1 – defines `Rule` as a subclass of `OntologyElement`. `OntologyElement` is defined in the OWL DL metamodel (Figure 3.1) as an element of an `Ontology`, via the composition link between `NamedElement` and `Ontology`. As can also be seen in Figure 3.1, the class `OntologyElement` is a subclass of the class `AnnotatableElement`, which defines that rules can be annotated. As annotations are modeled in the OWL DL metamodel, a URI reference can be assigned to a rule for identification.

A rule consists of an antecedent and a consequent, also referred to as the body and the head of the rule, respectively. Both the antecedent and the consequent consist of a set of atoms which can possibly be empty, as depicted by the multiplicity in Figure 4.1. Informally, a rule says that *if* all atoms of the antecedent hold, *then* the consequent holds. An empty antecedent is treated as trivially true, whereas an empty consequent is treated as trivially false.

The same antecedent or consequent can be used in several rules, as indicated in the metamodel by the multiplicity on the association between `Rule` on the one hand and `Antecedent` or `Consequent` on the other hand. Similarly, the multiplicities of the association between `Antecedent` and `Atom` and of the association between `Consequent` and `Atom` define that an antecedent and a consequent can hold zero or more atoms. The multiplicity in the other direction defines that the same atom can appear in several antecedents or consequents. According to the SWRL specifications, every `Variable` that occurs in the `Consequent` of a rule must occur in the `Antecedent` of that rule, a condition referred to as "safety"[1]..

4.2.2 Atoms, Terms and Predicate symbols

The atoms of the antecedent and the consequent consist of predicate symbols and terms. According to SWRL, they can have different forms:

- $C(x)$, where C is an OWL description and x an individual variable or an OWL individual[2], or C is an OWL data range and x either a data variable or an OWL data value[3];

- $P(x, y)$, where P is an OWL individual valued property and x and y are both either an individual

[1]
```
context Rule inv:
self.hasConsequent.containsAtom->term->forAll(t|
t.oclIsTypeOf(Variable) implies (self.hasAntecedent.containsAtom->term->exists(t))).
```
[2]
```
context Atom inv:
(self.hasPredicateSymbol.oclIsTypeOf(Class))
implies (self.term->size()=1
and ((self.term.oclIsTypeOf(IndividualVariable))
or (self.term.oclIsTypeOf(Individual)))).
```
[3]
```
context Atom:
(self.hasPredicateSymbol.oclIsTypeOf(DataRange))
implies ((self.term->size()=1)
and ((self.term.oclIsTypeOf(DataVariable))
or (self.term.oclIsTypeOf(DataValue)))).
```

variable or an OWL individual[4], or P is an OWL data valued property, x is either an individual variable or an OWL individual and y is either a data variable or an OWL data value[5];

- sameAs(x, y), where x and y are both either an OWL individual or an individual variable;
- differentFrom(x, y), where x and y are both either an OWL individual or an individual variable;
- builtIn$(r, x, ...)$, where r is a built-in predicate and x is a data variable or OWL data value[6].

  A builtIn atom could possibly have more than one variable or OWL data value.

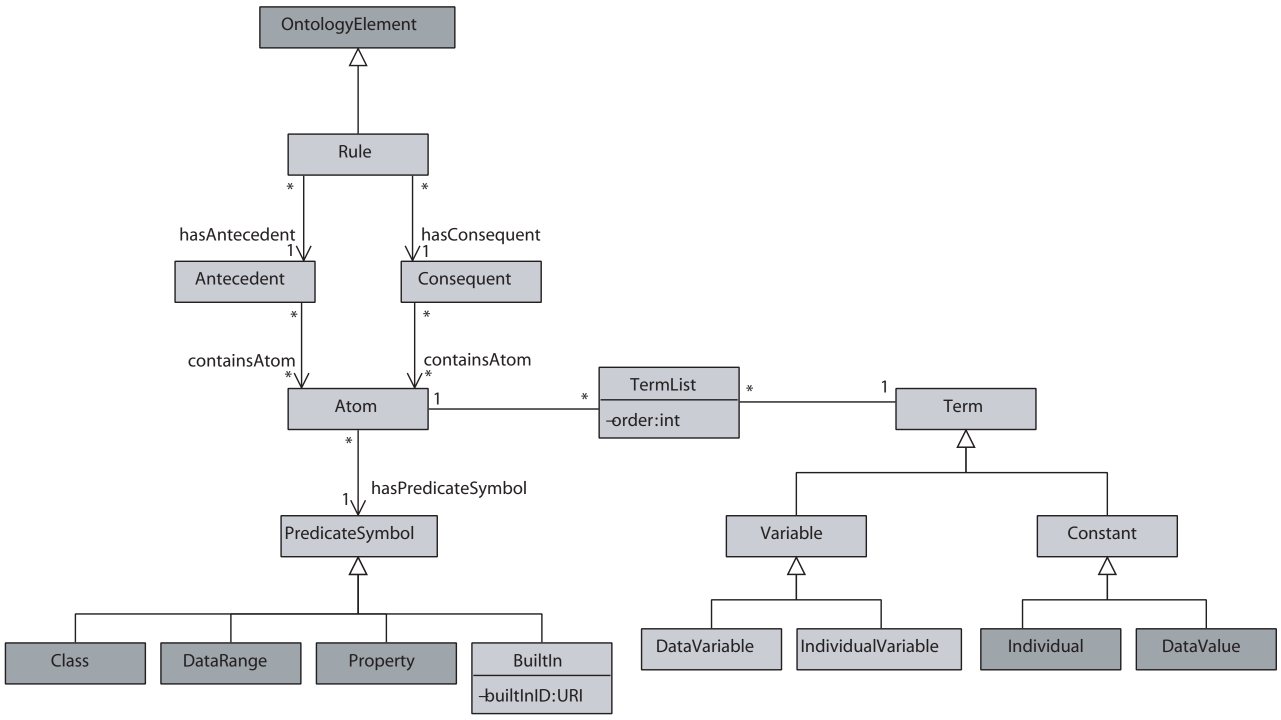

Figure 4.1: The Rule Definition Metamodel

The first of these—OWL description, data range, and property—were already provided in the OWL DL metamodel namely as metaclasses `Class`, `DataRange`, and `Property`, respectively. As can be seen in Figure 4.1, the predicates `Class`, `DataRange`, `Property`, and `BuiltIn` are all defined as subclasses of the class `PredicateSymbol`, which is associated to `Atom`. The remaining two atom types, `sameAs` and `differentFrom`, are represented as specific instances of `PredicateSymbol`.

To define the order of the atom terms, we put a class `TermOrder` in between `Atom` and `Term`. This UML association class connects atoms with terms and defines the term order via the attribute `order`.

[4]
```
context Atom inv:
(self.hasPredicateSymbol.oclIsTypeOf(ObjectProperty))
implies ((self.term->size()=2)
and (self.term->forAll((oclIsTypeOf(IndividualVariable))
or (oclIsTypeOf(Individual))))).
```
[5]
```
context Atom inv:
(self.hasPredicateSymbol.oclIsTypeOf(DatatypeProperty))
implies ((self.term->size()=2)
and ((self.termList.order=1) implies ((self.term.oclIsTypeOf(IndividualVariable)) or
(self.term.oclIsTypeOf(Individual))))
and ((self.termList.order=2) implies ((self.term.oclIsTypeOf(DataVariable))
or (self.term.oclIsTypeOf(DataValue))))) .
```
[6]
```
context Atom inv:
(self.hasPredicateSymbol.oclIsTypeOf(BuiltIn))
implies (self.term->forAll(oclIsTypeOf(DataVariable) or oclIsTypeOf(DataValue))).
```

Chapter 5

The Mapping Metamodel

5.1 Design Considerations

In contrast to the area of ontology languages, where the Web Ontology Language OWL has become a de facto standard for representing and using ontologies, there is no agreement yet on the nature and the right formalism for defining mappings between ontologies. In a recent discussion on the nature of ontology mappings, some general aspects of mapping approaches have been identified [SU05]. We briefly discuss these aspects in the following and clarify our view on mappings that is reflected in the proposed metamodel with respect to these aspects.

5.1.1 What do mappings define?

We here restrict our attention to declarative mapping specifications. In particular, we see mappings as axioms that define a semantic relation between elements in different ontologies.

A number of different kinds of semantic relations have been proposed. Most common are the following kinds of semantic relations:

Equivalence ($\equiv$) 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.

Containment ($\sqsubseteq$, $\sqsupseteq$) 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.

Overlap (o) 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. The corresponding relations can be used to describe that two elements are *not* equivalent ($\not\equiv$), *not* contained in each other ($\not\sqsubseteq$) or *not* overlapping or disjoint respectively (Ø). 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.

Extensional 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 In the case of intensional mappings, the semantic relations relate the elements directly, i.e. considering the properties of the element itself. In particular, if two elements are intensionally the same, they refer to exactly the same real world object.

5.1.2 What do mappings preserve?

It is normally assumed that mappings preserve the 'meaning' of the two models in the sense that the semantic relation between the intended interpretations of connected elements is the one specified in the mapping. A problem with this assumption is that it is virtually impossible to verify this property. Instead, there are a number of verifiable formal properties that mappings can be required to satisfy. Examples of such formal properties are the satisfiability of the overall model, the preservation of possible inferences or the preservation of answers to queries. Often, such properties can only be stated relative to a given application context, such as a set of queries to be answered or a set of tasks to be solved.

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 be the mapped ontologies. Generally, we can distinguish between a global domain and local domain assumption:

Global Domain It is assumed 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. 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.

Local Domains It is not assumed 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.

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. In such cases, e.g. in C-OWL [BGvH+03], the relationship is not defined directly as a set relationship between the extensions of the concepts, but can be defined in terms of domain relations that connect the interpretation domains by codifying how elements in one domain map into elements of the other domain.

Other assumptions made by approaches concerns the use of unique names for objects—this assumption is often made in the area of database integration—and the preservation of inconsistencies across mapped ontologies. In order to make an informed choice about which formalism to use, these assumptions have to be represented by the modeler and therefore need to be part of the proposed metamodel.

5.1.3 What do mappings connect?

In the context of this work, we decided to focus on mappings between ontologies represented in OWL DL. This restriction makes it much easier to deal with this aspect of ontology mappings as we can refer to the corresponding metamodel for OWL DL specified in [BH06a]. In particular, the metamodel contains the class `OntologyElement` 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.

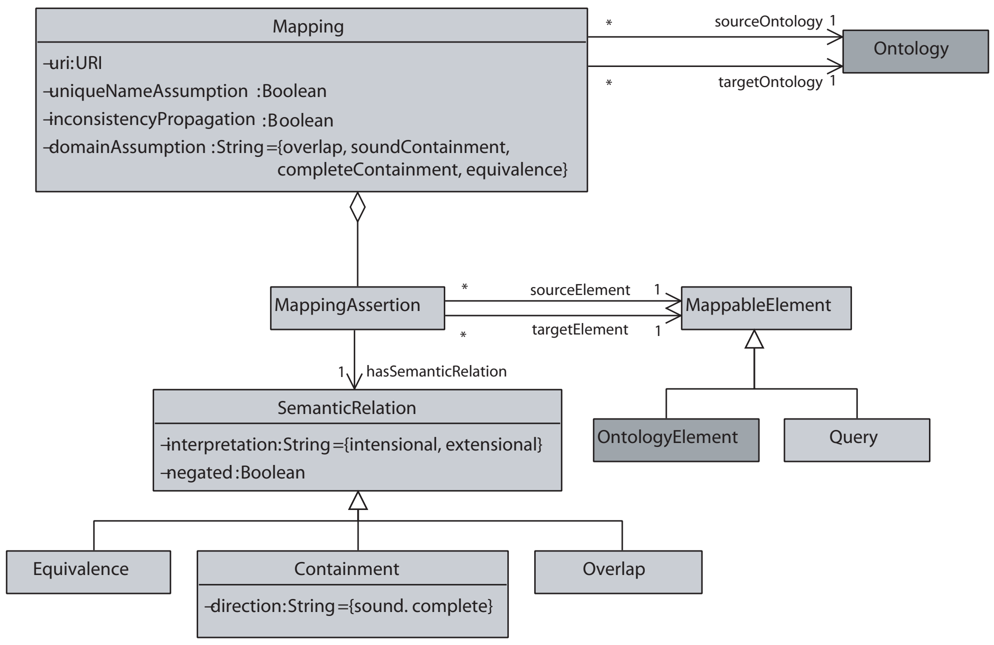
Mapping
-uri:URI
-uniqueNameAssumption :Boolean
-inconsistencyPropagation :Boolean
-domainAssumption :String={overlap, soundContainment, completeContainment, equivalence}
*
sourceOntology 1
*
targetOntology 1
Ontology
MappingAssertion
*
sourceElement 1
*
targetElement 1
MappableElement
1
hasSemanticRelation
SemanticRelation
-interpretation:String={intensional, extensional}
-negated:Boolean
OntologyElement
Query
Equivalence
Containment
-direction:String={sound. complete}
Overlap

Figure 5.1: Metamodel for ontology mappings

5.1.4 How are mappings organized?

The final question is how mappings are organized. They can either be part of a given model or be specified independently. In the latter case, the question is how to distinguish between mappings and other elements in the models. Mappings can be uni– or bidirectional. Further, it has to be defined whether a set of mappings is normative or whether it is possible to have different sets of mappings according to different applications, viewpoints or different matchers.

In this work, 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
- Mappings are first-class objects that exist independently of the ontologies. Mappings are directed and there can be more than one mapping between two ontologies

These choices leave us with a lot of freedom for defining and using mappings. Approaches that see mappings as parts of an ontology can be represented by the ontology and a single mapping. If only one mapping is defined between two ontologies, this can be seen as normative. Bi-directional mappings can be described in terms of two directed mappings.

5.2 A Metamodel for Ontology Mappings

We propose a metamodel for OWL ontology mappings as a consistent extension of the metamodels for OWL DL ontologies and rules, which supports mappings as described in Section 5.1. Constraints in OCL are defined to concretize the metamodel for a specific formalism. Like this, a constraint checker could for example check to which concrete formalism a certain UML model of mappings conforms.

Figure 5.1 shows the metamodel for mappings. In the figures, darker grey classes denote classes from the metamodels of OWL DL and rule extensions. The central class in the metamodel is the class `Mapping`,

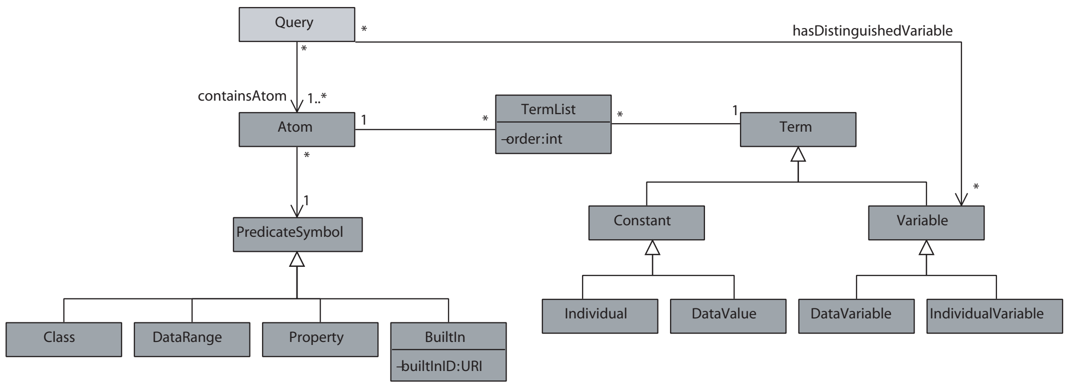

Figure 5.2: Metamodel for ontology mappings - definition of a query

having four attributes. The URI, defined by the attribute `uri`, allows to uniquely identify a mapping and refer to it as a first-class object.

A mapping is always defined between two ontologies. An ontology is represented by the class `Ontology` in the OWL DL metamodel. Two associations from `Mapping` to `Ontology`, `sourceOntology` and `targetOntology`, specify the source respectively the target ontology of the mapping. Cardinalities on both associations denote that to each `Mapping` instantiation, there is exactly one `Ontology` connected as source and one as target.

A mapping consists of a set of mapping assertions, denoted by the MOF aggregation 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.

In Section 5.1.1, we have introduced four semantic relations along with their logical negation to be defined in the metamodel. Two of these relationship types are directly contained in the metamodel through the subclasses `Equivalence` and `Overlap` of the class `SemanticRelation`. The other two, containment in either direction, are defined through the subclass `Containment` and its additional attribute `direction`, which can be `sound` ($\sqsubseteq$) or `complete` ($\sqsupseteq$).

The negated versions of all semantic relations are specified through the boolean attribute `negated` of the class `SemanticRelation`. For example, a negated `Overlaps` relation specifies the disjointness of two elements. The other attribute of `SemanticRelation`, `interpretation`, defines whether the mapping assertion is assumed to be interpreted intentionally or extensionally. Please note that the metamodel in principle supports all semantic relations for all mappable elements, including individuals.

As discussed in Section 5.1, a mapping assertion can connect two mappable elements, which may be ontology elements or queries. To support this, `MappableElement` has two subclasses `OntologyElement` and `Query`. The former is previously defined in the OWL DL metamodel. The class `Query` reuses constructs from the SWRL metamodel. The reason for reusing large parts of the rule metamodel lies in the fact that conceptually 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. Figure 5.2 shows this connection and shows how a `Query` is composed. They depict how atoms from the antecedent and the consequent of SWRL rules can be composed. Similarly, a `Query` also contains a `PredicateSymbol` and some, possibly just one, `Term`s. We defined the permitted predicate symbols through the subclasses `Class`, `DataRange`, `Property` and `BuiltIn`. Similarly, the four different types of terms are specified as well. The UML association class

`TermList` between `Atom` and `Term` allows to identify the order of the atom terms. Distinguished variables of a query are differentiated through an association between `Query` and `Variable`.

5.2.1 OCL Constraints for C-OWL

As explained before, we define OCL constraints on the mapping metamodel to concretize it according to specific formalism. This section lists the constraints for the C-OWL formalism.

- C-OWL does not have unique name assumption:

  context Mapping inv:
  self.uniqueNameAssumption = false

- C-OWL does not have inconsistency propagation:

  context Mapping inv:
  self.inconsistencyPropagation = false

- The domain assumption is always overlap:

  context Mapping inv:
  self.DomainAssumption = 'overlap'

- Mappings can not be defined between queries; mappable elements are object properties, classes and individuals:

  context MappableElement inv:
  self.oclIsTypeOf(ObjectProperty) or
  self.oclIsTypeOf(Class) or
  self.oclIsTypeOf(Individual)

- Elements being mapped to each other, must be of the same kind:

  context MappingAssertion.sourceElement.oclType : OclType
  body MappingAssertion.targetElement.ocltype

- C-OWL supports the following semantic relations: equivalence, containment (sound as well as complete), overlap and negated overlap (disjoint):

  context SemanticRelation inv:
  (self.oclIsTypeOf (Equivalence) and self.negated = false) or
  (self.oclIsTypeOf(Containment) and self.negated = false) or
  self.oclIsTypeOf(Overlap)

5.2.2 OCL Constraints for DL-Safe Mappings

As for C-OWL, for DL-Safe Mappings we also define OCL constraints which specialize the abstract metamodel in the first part of this chapter. We list the constraints below.

- DL-safe mappings have no unique name assumption:

  context Mapping inv:
  self.uniqueNameAssumption = false

- DL-safe mappings have inconsistency propagation:

  context Mapping inv:
  self.inconsistencyPropagation = true

- The domain assumption is always equivalence:

  context Mapping inv:
  self.domainAssumption = 'equivalence'

- Mappable Elements are queries, Properties, Classes, Individuals and DataRange:

  context MappableElement inv:
  self.oclIsTypeOf(Query) or
  self.oclIsTypeOf(Property) or
  self.oclIsTypeOf(Class) or
  self.oclIsTypeOf(Individual) or
  self.oclIsTypeOf(DataRange)

- Elements being mapped to each other, must be of the same kind (this means, if someone wants to map e.g. a concept to a query, the concept should be modelled as a query):

  context MappingAssertion.sourceElement.oclType : OclType
  body MappingAssertion.targetElement.oclType

- If queries are being mapped to each other, they must have the same distinguished variables:

  context MappingAssertion inv:
  (self.sourceElement.oclIsTypeOf(Query)
  and self.targetElement.oclIsTypeOf(Query))
  implies (self.sourceElement.hasDistinguishedVariable
  = self.targetElement.hasDinstinguishedVariable)

- In DL-safe mappings, interpretation is always extensional:

  context SemanticRelation inv:
  self.interpretation = ÔextensionalÕ

- DL-safe mappings support the following semantic relations: equivalence and containment (sound as well as complete):

  context SemanticRelation inv:
  (self.oclIsTypeOf(Equivalence) and self.negated = false) or
  (self.oclIsTypeOf(Containment) and self.negated = false)

Chapter 6

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.

6.1 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].

6.1.1 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:

- $i : \phi \xrightarrow{\sqsubseteq} j : \psi$ (*into* rule), with semantics: $r_{ij}(\phi^{I_i}) \subseteq \psi^{I_j}$
- $i : \phi \xrightarrow{\sqsupseteq} j : \psi$ (*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 $i : \phi \xrightarrow{\sqsubseteq} j : \psi$ is not equivalent to $j : \psi \xrightarrow{\sqsupseteq} i : \phi$.

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 $q_1 \wedge ... \wedge q_m$, 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, 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.

6.1.2 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 rely 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 provide 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.

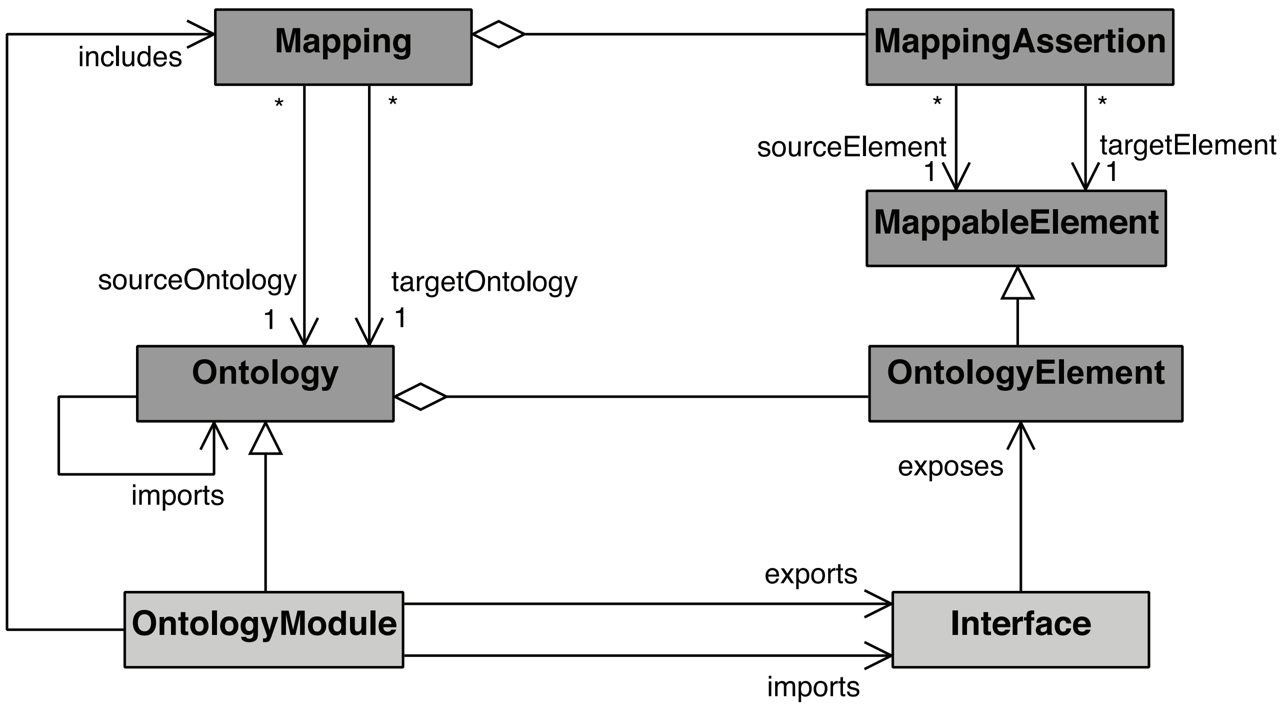

Figure 6.1: Metamodel extensions for ontology modules

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.

6.2 A Metamodel for Modular Ontologies

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 DL ontologies and mappings.

Figure 6.1 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 an export interface and a set of import interfaces. The export interface, modeled via the `exports` association, `exposes` the set of `OntologyElement`s that are intended to be reused by other modules.

The reuse of a module by another module is modeled via the regular `imports` relationship already provided by the class `Ontology`. To each imported ontology module is associated an import interface, modeled through the `imports` association, that `exposes` the `OntologyElement`s the importing module reuse from the imported module. Additionally, a `Module` also provides an `imports` relationship with the `Mapping` class, which is used to relate different ontology modules via ontology mappings.

6.2.1 Abstract Syntax for Ontology Modules

The goal of this section is to come up with an abstract syntax for the ontology modularization. The purpose of this is to identify the necessary information to be accommodated in an ontology module as well as structural properties of a modularized networked ontology. This will be done on a solid formal basis which will enable us to define a corresponding semantics at a later stage.

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, URIs 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.

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 language NAMED ELEMENTS.
  In the case of OWL, Nam will be thought to contain all class names, role 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 $L : 2^{Nam} \to 2^{Elem}$ 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 constructs[1],
- For a given set O of ontology axioms, let $\mathrm{Sig}(O)$ 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 2 *An* ONTOLOGY MODULE $\mathcal{OM}$ *is a tuple* $\langle id, Imp, \mathcal{I}, M, O, E\rangle$ *where*

- $id \in Id_{\mathrm{Modules}}$ *is the identifier of* $\mathcal{OM}$
- $Imp \subseteq Id_{\mathrm{Modules}}$ *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 the 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* $\mathcal{OM}$*),*
- $M \subseteq Id_{\mathrm{Mappings}}$ *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),*
- $E \subseteq \mathrm{Sig}(O) \cup \bigcup_{id \in Imp} \mathcal{I}_{id}$ *is called* EXPORT INTERFACE *(telling which named entities from the ontology module are "published", i.e., can be imported by other ontology modules).*

As a simple example let us consider an ontology module $\mathcal{OM}_i$ (with module identifier i) completely importing another ontology module $\mathcal{OM}_k$ (with module identifier k) and exporting everything:
$\mathcal{OM}_i = \langle i, \{k\}, \{\mathrm{Sig}(O_k)\}, \emptyset, O_i, \mathrm{Sig}(O_i) \cup \mathrm{Sig}(O_k)\rangle$
In a further step we formally define the term mapping (which is supposed to be a directed link between two ontology modules establishing semantic correspondences between them).

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

Definition 3 *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 TYPES[2]

To finalize the definitional part, it remains to formally establish the connection between the ontology modules and mappings and their identifiers. So we define the *module space* which abstractly reflects the URI reference structure of the Web.

Definition 4 *A* MODULE SPACE *is defined as pair* $\langle \mathfrak{Mod}, \mathfrak{Map}\rangle$ *where*

- $\mathfrak{Mod} = \{\mathcal{OM}_{id}\}_{id \in Id_{\text{Modules}}}$ *is a set of ontology modules (each endowed with a unique module identifier) and*
- $\mathfrak{Map} = \{M_{\text{mid}}\}_{\text{mid} \in Id_{\text{Mappings}}}$ *is a set of mappings (with associated unique mapping identifiers).*

We conclude this section by defining additional requirements to modules and module spaces which will be assumed in the sequel:

We call a module space IMPORT-EXPORT-COMPATIBLE, if for every $\mathcal{OM}_{id_1} = \langle id_1, Imp, \mathcal{I}, M, O, E\rangle \in \mathfrak{Mod}$ and every $id_2 \in Imp$ with $\mathcal{OM}_{id_2} = \langle id_2, Imp', \mathcal{I}', M', O', E'\rangle$ we have that $I_{id_2} \subseteq E'$. Import-export-compatibility just states that every primitive being imported from a module must be exposed in that module's export.

For a given $id \in Id_{\text{Modules}}$, we call the module $\mathcal{OM}_{id} = \langle Imp, \mathcal{I}, M, O, E\rangle$ of a module space $\langle \mathfrak{Mod}, \mathfrak{Map}\rangle$ MAPPING-COMPATIBLE, if for every $\text{mid} \in M$ with $M_{\text{mid}} = \langle s, t, C\rangle$ we have that $s, t \in Imp \cup \{id\}$. Mapping compatibility demands that if a mapping is imported into a module, both source module and target module of that mapping must be imported as well (or be the importing module itself).

[2] In accordance with the NeOn metamodel, this set will be fixed to $R = \{\sqsubseteq, \sqsupseteq, \equiv, \perp, \not\sqsubseteq, \not\sqsupseteq, \not\equiv, \not\perp\}$

Part III

Metadata for Networked Ontologies

Chapter 7

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.

7.1 Preliminary considerations

7.1.1 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.

7.2 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.

7.3 OMV - 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.

7.3.1 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.

7.3.2 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 means to distinguish between required and optional metadata properties. The implementation was performed manually using a common ontology editor.
The main classes and properties of the OMV ontology are illustrated in Figure 7.1. [1]

7.3.3 OMV core metadata entities

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 `Person`s 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 Implementation`s. Review details and further information can be captured in an extensional OMV module. Further on we

[1] Please notice, that not all classes and properties are included. The ontology is available for download in several ontology formats at `http://ontoware.org/projects/omv/`

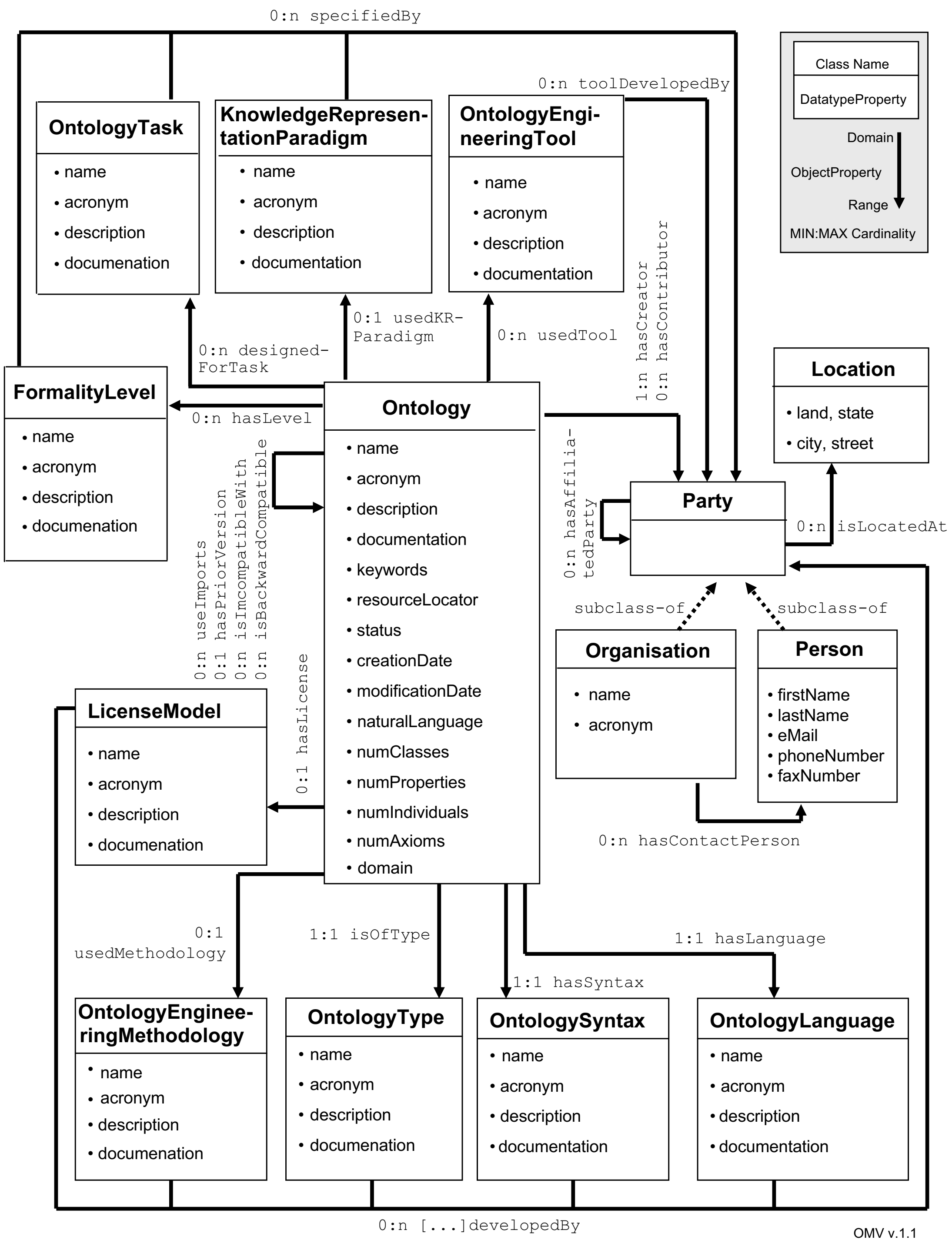

Figure 7.1: OMV overview

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 `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 `numClasses`, `numProperties`, `numAxioms`, `numIndividuals`.

Chapter 8

Conclusion

8.1 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 lifecycle of such a new generation of semantic applications. In this deliverable we have presented a 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 developed four modules of the metamodel:

1. The OWL metamodel serves as the core of our networked ontology model,
2. the rule metamodel extends the ontology language with the expressiveness to model horn-like rules,
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.

The specification of the metamodel only is a first step towards the final goal of realizing a reference architecture for semantic applications based on networked ontologies. In the following we discuss future steps to be taken.

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

8.2 Roadmap for Future Work

Future Work will include:

1. Extensions and refinements of the current metamodel to deal with additional aspects of networked ontologies, such as an explicit representation of context, and the alignment with ongoing efforts for the standardization of ontology languages.
2. The implementation of components in the NeOn toolkit to support the individual phases of the lifecycle of networked ontologies, including modeling, maintaining, and evolving networked ontologies.

The extensions and refinements will be the focus of the work on the subsequent Deliverable D1.1.2 *Networked Ontology Model, Updated Version*. In particular, we will continue the refinement of the language for modular ontologies and work on extensions for versioning. At the same time, we will monitor ongoing standardization efforts around OWL 1.1 and the rule language to be developed in the RIF working group to align our networked ontology model with these developments.

Several upcoming deliverables as part of WP1 will complement the networked ontology model with methods and techniques for *Consistency Models for Networked Ontologies* (D1.2.1), *Change Propagation for Networked Ontologies* (D1.3.1) and support for the *Management of Networked Ontologies* (D1.4.1). The corresponding tool support will be integrated into the NeOn toolkit as part of the WP6 activities. The first version of this toolkit will be available by M12.

The next steps for OMV include the development of OMV extension ontologies elaborating aspects for specific domains, tasks or communities. For this task we envision the collaborations and contributions from NeOn partners. Additionally, future work includes the evaluation of the application of OMV in different scenarios, pushing OMV to a community standard and the refinement of the current OMV core.

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 are capitalized. If the instance name contains more than one word, each word is separated by a blank space and capitalize each word. I.e. "Task Ontology" "Highly Informal"

A.2 Prefix conventions

The use of prefix conventions helps to identify in an easy way ontology elements. NeOn uses prefix conventions to distinguish **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.

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)*. World Wide Web Consortium, September 2005. W3C Member Submission, `http://www.w3.org/Submission/WRL/`.

[BCH06a] Jie Bao, Doina Caragea, and Vasant 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] Jie Bao, Doina Caragea, and Vasant Honavar. Modular ontologies - a formal investigation of semantics and expressivity. In *R. Mizoguchi, Z. Shi, and F. Giunchiglia (Eds.): Asian Semantic Web Conference 2006, LNCS 4185*, pages 616–631, 2006.

[BCH06c] Jie Bao, Doina Caragea, and Vasant 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] Jie Bao, Doina Caragea, and Vasant 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.

[BGvH$^+$03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Second International Semantic Web Conference ISWC'03*, volume 2870 of *LNCS*, pages 164–179. Springer, 2003.

[BH06a] Saartje Brockmans and Peter Haase. A Metamodel and UML Profile for Networked Ontologies – A Complete Reference. Technical report, Universität Karlsruhe, April 2006. `http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/ontology-metamodeling.pdf`.

[BH06b] Saartje Brockmans and Peter Haase. A Metamodel and UML Profile for Rule-extended OWL DL Ontologies –A Complete Reference. Technical report, Universität Karlsruhe, March 2006. `http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/owl-metamodeling.pdf`.

[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 *4th Int. Conf. on UML (UML 2001)*, Toronto, Canada, October 2001.

[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] Alexander Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *J. Data Semantics*, 1:153–184, 2003.

[CP99] S. Cranefield and M. Purvis. UML as an Ontology Modelling Language. In *Proceedings of the Workshop on Intelligent Information Integration*, volume 23 of *CEUR Workshop Proceedings*, Stockholm, Sweden, July 1999.

[dSM06] Mathieu d'Aquin, Marta Sabou, and Enrico Motta. Modularization: a key for the dynamic selection of relevant knowledge components. In *Workshop on Modular Ontologies*, 2006.

[Fow03] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[GKM05] B. Grosof, 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, 27-28 April 2005, Washington, DC, USA*, 2005.

[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.

[GPSK05] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Automatic partitioning of owl ontologies using -connections. In *Description Logics*, 2005.

[HEC$^+$04] Lewis Hart, Patrick Emery, Bob Colomb, Kerry Raymond, Sarah Taraporewalla, Dan Chang, Yiming Ye, and Mark Dutra Elisa Kendall. OWL full and UML 2.0 compared, March 2004. `http://www.itee.uq.edu.au/$\sim$colomb/Papers/UML-OWLont04.03.01.pdf`.

[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 *Lecture Notes in Computer Science*, pages 37–41. Springer, 2005.

[HPSB$^+$04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. Technical report, May 2004. W3C Member Submission, `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`.

[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.

[IBM05] IBM, Sandpiper Software. *Ontology Definition Metamodel, Fourth Revised Submission to OMG*, November 2005.

[KLWZ03] Oliver Kutz, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. E-connections of description logics. In *Description Logics Workshop, CEUR-WS Vol 81*, 2003.

[Kre98] R. Kremer. Visual Languages for Knowledge Representation. In *Proc. of 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Voyager Inn, Banff, Alberta, Canada, April 1998. Morgan Kaufmann. `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/kremer/`.

[LTGP04] A. Lozano-Tello and A. Gomez-Perez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2), 2004.

[MHRS06] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can owl and logic programming live together happily ever after? In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006.

[MKW04] S. J. Mellor, S. Kendall, and A. Uhl D. Weise. *MDA Distilled.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[MSS04] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *International Semantic Web Conference*, pages 549–563, 2004.

[Mv03] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), August 2003. Internet: http://www.w3.org/TR/owl-features/.

[Obj02] Object Management Group. Meta Object Facility (MOF) Specification. Technical report, Object Management Group (OMG), April 2002. `http://www.omg.org/docs/formal/02-04-03.pdf`.

[Org04] National Information Stadards 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] Heiner Stuckenschmidt and Michel C. A. Klein. Integrity and change in modular ontologies. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 900–908, 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] Julian Seidenberg and Alan Rector. Web ontology segmentation: Analysis, classification and use. In *Proceedings of the World Wide Web Conference (WWW)*, Edinburgh, June 2006.

[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 - IJCAIÂě05*, 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 Tamilin. 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] Heiner Stuckenschmidt and Michael Uschold. Representation of semantic mappings. In Yannis Kalfoglou, Marco Schorlemmer, Amit Sheth, Steffen Staab, and Michael Uschold, editors, *Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings*, volume 04391, Germany, 2005. IBFI, Schloss Dagstuhl.

[TF05] Sergio Tessaris and Enrico Franconi. Rules and queries with ontologies: a unifying logical framework. In *Description Logics*, 2005.

[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.

[Vol04] R. Volz. *Web Ontology Reasoning with Logic Databases*. Phd thesis, University of 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, 27-28 April 2005, Washington, DC, USA*, 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.

[WK04] Jos Warmer and Anneke Kleppe. *Object Constraint Language 2.0*. MITP Verlag, 2004.