NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D3.1.3 Improved NeOn formalism for context representation

Deliverable Co-ordinator: **Guilin Qi**

Deliverable Co-ordinating Institution: **UKarl**

Other Authors: **Peter Haase**

This deliverable provides a common formalism for representing context of ontologies. We follow the generic definition of context as *modifiers of semantics* introduced in our previous work. As a main contribution, we provide a syntax compatible with the Networked Ontology Model and the OWL 1.1 ontology language, which enables the representation of (arbitrary) context information within an ontology. We provide three instantiations of specific context languages based on the abstract definition and the common representation syntax. These three forms of context cover *Provenance*, *Argumentation* and *Mapping* information.

Document Identifier:	NEON/2007/D3.1.3/1.0	Date due:	February 29, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 29, 2008
Project start date	March 1, 2006	Version:	1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

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

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 Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**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: Jesús Contreras E-mail address: jcontreras@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 E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
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	**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	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

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 to the writing of this document or its parts:

- UKARL
- INRIA
- JSI

Change Log

Version	Date	Amended by	Changes
0.1	10-11-2007	Guilin Qi	Create the Deliverable
0.2	10-01-2007	Peter Haase	Revise Chapter 3 and Chapter 4
0.3	12-01-2007	Guilin Qi	Revise Chapter 4
0.4	29-01-2007	Peter Haase and Guilin Qi	Revise Chapter 3 and Chapter 4
0.5	31-01-2007	Peter Haase and Guilin Qi	Final Write-up
0.6	05-03-2007	Peter Haase and Guilin Qi	Final Revision by Incorporating Review's Comments

Executive Summary

Real life ontologies and corresponding data are produced by individuals or groups in certain settings for specific purposes. Because of this, they can almost never be considered as something absolute in their semantics, but instead can be viewed as valid in a certain *context*. In deliverable D3.1.1 we introduced a generic and abstract definition of context as *modifiers of semantics* and surveyed a number of different approaches to dealing with context along this generic definition. In NeOn deliverable D3.1.2, we instantiated the abstract definition for specific types of context relevant for NeOn.

In this deliverable we continue our previous work on the NeOn formalisms for context representation. We first address the question how context can be syntactically represented to be able to relate an ontology with its context. Our approach is directly built on the networked ontology model and is compatible with the OWL 1.1 language, support for which is provided in the NeOn infrastructure. In terms of our generic definition of context, we provide a common representation for the knowledge language and the context language. With our proposed formalism, we enable representing arbitrary context information itself in the form of an OWL ontology.

We then instantiate our generic definition of context to three specific forms of context: *Provenance*, *Argumentation* and *Mapping*. Provenance is a form of context that is typically available for automatically generated ontologies – e.g. in ontology learning; Argumentation Structures are a form of context that is obtained in collaborative ontology engineering processes; Mappings are a form of context that is obtained by ontology mapping systems to achieve an interoperation between applications or data relying on different ontologies. For each of these types of context, we gave an example to show how it can be represented by the syntactical representation. Finally, we discuss the semantics of the three types of context by relating them to our abstract definition of context.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 NeOn Big Picture

Real life ontologies and corresponding data are produced by individuals or groups in certain settings for specific purposes. Because of this, they can almost never be considered as something absolute in their semantics and are often inconsistent with ontologies created by other parties under other circumstances. In order to fully utilize networked ontologies, those disagreements must be identified prior to using them for reasoning. Each ontology can be viewed as valid (or appropriate) in a certain context. We are interested in knowledge expressed as a set of assertions and rules. If such a set of assertions is put into a context, then this means that the context alters some of the meaning of the set of assertions. From the theoretical side, we could say that whenever the contextual information is necessary, the target ontology cannot have fully defined static semantics because it depends on some external information which we call *context*. We could call such ontologies *parametric ontologies* because their semantics depends on the value of contextual *parameters*. It is the goal of the work performed in WP3 to develop appropriate techniques for dealing with context. As shown in Figure 1.1, this work belongs to the central part of the research and development WPs in NeOn. One of the key points of this workpackage is to model and provide a formalization of the context. This model will support both a proper representation of the information particular to the context and its formalization that allows reasoning with the modeled context.

1.2 Context Representation in NeOn

The notion of context has a very long history within several research communities, leading to vast studies about how to define a context, how to take into account information coming from the context, how to contextualize or de-contextualize knowledge and information, etc.

In D3.1.1 we surveyed the state-of-the-art on dealing with context. We identified several possible usages of context for ontologies and gave an overview of some present approaches for representing and reasoning with context which may be relevant for NeOn. We provided an abstract and generic mathematical definition of context, based on which we compared the different approaches. In terms of the usages of context, we found that *supporting viewpoints and perspectives* and *dealing with inconsistent, uncertain and vague information*, will play a paramount role in NeOn. To be able to address these usage scenarios for context, we identified that the following approaches for contexts are relevant for NeOn: The *networked ontology model* developed in WP1 provides the most obvious form of context: Ontologies will be embedded in a network of ontologies, which forms the context for its interpretation. In the networked scenarios of NeOn, ontologies are not treated as isolated entities, but are related to other ontologies in various networked ways, including versioning and mapping information etc. These other ontologies together with these links can be understood as a context for the ontology, as they will (in some cases) alter the knowledge which can be inferred from the ontology. *Reasoning with inconsistent ontologies* exploiting context information is important when different information sources with contradicting information will be integrated. Contextual information can be used to

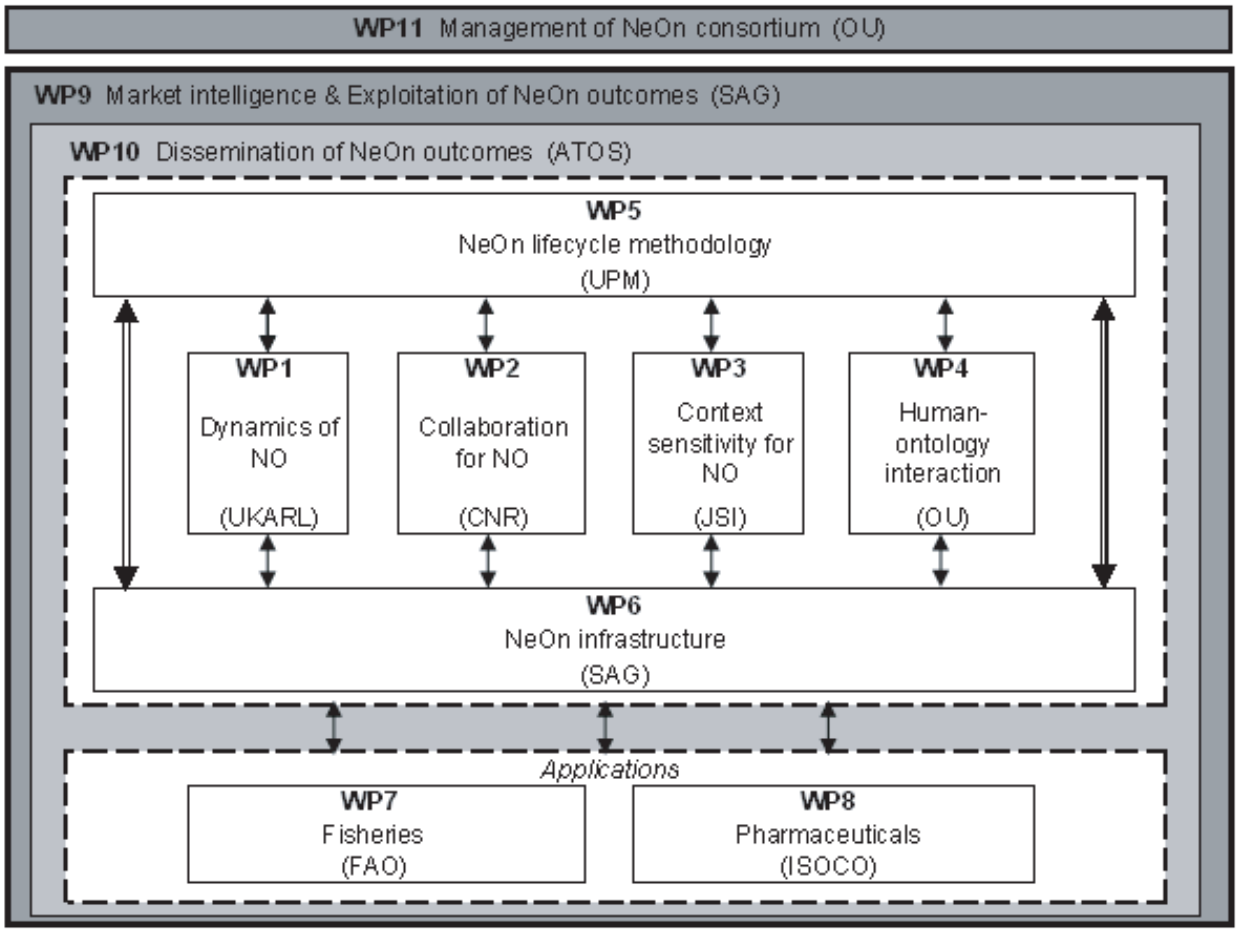

Figure 1.1: Relationships between different workpackages in NeOn

resolve such conflicts. It can be used to select relevant consistent parts of the knowledge base which suffice for the task at hand. Contextual information provides guidance for this selection process, as usually different possibilities exist for resolving an inconsistency. For example, in NeOn deliverable D1.2.2, we apply the confidence values which are provenance information to resolve inconsistency and incoherence. *Context-based selection functions* appear promising for addressing a number of different problems. Finally, a combination of *possibilistic* and *probabilistic logics* seems to be required to deal with the various forms of vagueness and uncertainty in a contextualized way. The numerical values or priority information attached to elements of an ontology will provide important context information to deal with imprecision in an ontology.

Considering these findings, in this deliverable we define the NeOn formalism for context representation. We first recall the generic definition of context. We then consider how the context can be syntactically represented to be able to relate an ontology with its context and propose so called groundlings of the context representation within OWL that allow to specify the context itself in the form of an OWL ontology. Finally, we instantiate our generic definition of context for three specific forms of context: *Provenance*, *Arguments* and *Mapping*. For example, in a provenance ontology, each of the ontology elements is associated with a confidence value, a relevance value, and some provenance elements that can be used to generate formal or informal explanations for particular results. We also illustrate how the specific types of context can be represented by the context ontologies.

The work given in this deliverable is closely related to other activities in WP3. In deliverable D3.8.1, several ontology learning tools are presented. These tools can automatically generate ontologies attached with provenance information. In deliverable D3.3.1, the Alignment Server is given as the infrastructure for con-

textualizing ontologies by finding relations that it has with other ontologies. In our deliverable, we provide a common metamodel and concrete syntax for the mapping generated by the Alignment Server and ontology mediation module given in deliverable D3.2.1. We give an example to illustrate how the grounded ontology in deliverable D3.2.1 can be represented by the model used to represent provenance context.

1.3 Overview of the Deliverable

In this deliverable we start with a generic and abstract definition of context in Chapter 2. This definition was already used in the state-of-the-art deliverable on context representation languages D3.1.1 [HHR+06]. We provide an OWL-based syntax for context in Chapter 3. In Chapter 4 we then present instantiations of this generic definition for a number of types of contexts that we identified as particularly interesting for the NeOn project. We conclude with a roadmap for future work in Chapter 5.

Chapter 2

A Generic Definition for Context

In this chapter we recapitulate the formal definition of *context of an ontology*[1], which we first provided in D3.1.1 [HHR+06]. This definition is not intended to be operationalized in a reasoner. The purpose of this definition rather is to provide a basis for a characterization and comparison of different context representation formalisms within a common framework by instantiating the generic definition.

Contexts as modifiers of semantics. We are interested in knowledge expressed as a set of assertions and rules. Examples are knowledge bases (or ontologies), and relations between such knowledge bases.

If such a set of assertions is put into a context, then this means that the context alters some of the meaning of the set of assertions. In other words, the context acts as a *modifier* for the semantics of a knowledge base.

2.1 Formal abstract definition of context for an ontology

Let K be a knowledge base, which comes with an associated semantics $S(K)$. Thus, S is a function which associates a semantics to any knowledge base K.

Now, given a context C and a knowledge base K, we denote by $S'(K, C)$ the semantics of K *in the context* C. Thus, S' is a function which associates to any knowledge base K and context C a semantics, e.g. expressed by the set of all logical consequences of K in the context C.

If we have empty context (denoted by $\emptyset$), then often we require $S'(K, \emptyset) = S(K)$.

Note that there is a convenient way to describe the function S' in many cases. Given a knowledge base K and context C, it will often be possible to create a knowledge base K' such that $S'(K, C) = S(K')$. In these cases, reasoning within a context can be reduced to changes of the knowledge base K (converting it into K'), and by reusing existing reasoners.

Formal Definition of Context We will now go into further detail. Taking some language L, a *knowledge base on* L is a (possibly infinite) set of statements over L. Let $\mathcal{KB}(L)$ denote the set of all knowledge bases expressible in L.

Now, we consider a language L_{k} called *knowledge language*. A semantics for L_{k} can be formalized as a function $S : \mathcal{KB}(L_{\mathrm{k}}) \rightarrow \mathcal{KB}(L_{\mathrm{k}})$ assigning to a knowledge base K a knowledge base $S(K)$ containing all logical consequences of K expressible in the knowledge language.

Let furthermore be L_{c} a language called *context language* for expressing contextual knowledge. An L_{c}-context semantics for L_{k} is then a function $S : \mathcal{KB}(L_{\mathrm{k}}) \times \mathcal{KB}(L_{\mathrm{c}}) \rightarrow \mathcal{KB}(L_{\mathrm{k}})$. (The overloading of the symbol S is by purpose.)

[1] Please note that within this deliverable we do not distinguish between the use of the notion of "ontology" and that of "knowledge base". However, for the logical characterization, we tend to prefer the term "knowledge base".

In practice, one will mostly impose further restrictions on the knowledge base one works with. E.g., a knowledge base could be required to contain only certain kinds of expressions from L_{k} – as an easy example, take a database containing only tuples of entities (or, similarly, a logic program containing only ground facts) while the entailed knowledge (respectively the expressible queries) could have a much more complex structure. Another common constraint to knowledge bases is that they have to be finite (or at least finitely representable in some sense). The set of finite knowledge bases over some language L_{k} will be denoted by $\mathcal{KB}_{\mathrm{fin}}(L_{\mathrm{k}})$.
In many cases, additional constraints will be reasonable. In particular, we will call a context semantics

- *conservative*, if $S(K, \emptyset) = S(K)$ for all $K \in \mathcal{KB}(L_{\mathrm{k}})$. This means that, if an empty context (i.e. no contextual information) is provided, the semantics coincides with the "pure" semantics of the knowledge language.
- *extensive*, if $K \subseteq S(K, C)$ for all $K \in \mathcal{KB}(L_{\mathrm{k}})$ and for all $C \in \mathcal{KB}(L_{\mathrm{c}})$, i.e., all statements of the knowledge base are as well logical consequences of it. In other words, any information stated in the knowledge base can be deduced to be valid (and cannot be spoilt by whatever context provided).
- *knowledge-monotone*, if $K_1 \subseteq K_2$ implies $S(K_1, C) \subseteq S(K_2, C)$ for all $K_1, K_2 \in \mathcal{KB}(L_{\mathrm{k}})$ and $C \in \mathcal{KB}(L_{\mathrm{c}})$, i.e., all logical consequences remain valid if the knowledge base is augmented and the context does not change. Note, that this is not always the case (cf. non-monotonic semantics by closed world assumption).
- *context-monotone*, if $C_1 \subseteq C_2$ implies $S(K, C_1) \subseteq S(K, C_2)$ for all $C_1, C_2 \in \mathcal{KB}(L_{\mathrm{c}})$ and $K \in \mathcal{KB}(L_{\mathrm{k}})$, i.e., if the information given by the context increases, the derivable information does so as well. In particular, no previously valid consequence can be invalidated by adding more contextual knowledge.
- *idempotent*, if $S(S(K, C), C) = S(K, C)$ for all $C \in \mathcal{KB}(L_{\mathrm{c}})$ and $K \in \mathcal{KB}(L_{\mathrm{k}})$, i.e., taking all consequences of a knowledge base under a certain context and then taking again all consequences under the same context will yield nothing new.
- *dependently reducible*, if there is a function $\sigma : \mathcal{KB}(L_{\mathrm{k}}) \times \mathcal{KB}(L_{\mathrm{c}}) \to \mathcal{KB}(L_{\mathrm{k}})$, such that $S(K, C) = S(\sigma(K, C), \emptyset)$, i.e., knowing a knowledge base K and a context C, one can determine a new finite knowledge base with the same set of consequences as K with context C. I.e. for every contextualized knowledge base we can determine a logically equivalent knowledge base without context.
- *independently reducible*, if there is a function $\tau : \mathcal{KB}(L_{\mathrm{c}}) \to \mathcal{KB}(L_{\mathrm{k}})$ such that $S(K, C) = S(K \cup \tau(C), \emptyset)$ for all $K \in \mathcal{KB}(L_{\mathrm{k}})$ and $C \in \mathcal{KB}(L_{\mathrm{c}})$, i.e., any context can be "translated" into L_{k} (independently from K) and simply added to the knowledge base. In this case, contextual reasoning could be reduced to pure reasoning over L_{k}, such that existing methods could easily be employed for this.

The above definition is very abstract. This is done on purpose to accommodate the many practically important ways of context usage. In Chapter 4, we give some examples of concrete instances of the abstract definition. Many more notions of context fit our general definition. In the project, we will have to determine which concrete instances will be used and supported by the NeOn system. These instances will have to be dealt with on an individual basis when realizing the NeOn system.

Chapter 3

Representing Context in OWL Ontologies

In this chapter we address the question how context can be syntactically represented so as to relate an ontology to its context. In terms of our generic definition of context, we provide a common representation for the knowledge language L_{k} and the context language L_{c}.

Within NeOn, the knowledge language is already specified by the networked ontology model [HBP+07], defining OWL as the primary knowledge language. Yet, for the context language L_{c}, no commonly accepted representation formalism exists. With our proposed formalism, we enable representing arbitrary context information itself in the form of an OWL ontology. It should be noted, that this approach is only targeting the *syntactic representation* of the context in OWL, but does not attempt to capture the *semantics* of the context languages within the OWL language. The semantics of the context language can be instantiated by specific forms of context following the generic definition from Chapter 2.

3.1 Requirements and Difficulties

Without going into the intricacies of specific context languages (as we will do in the subsequent chapter), we here present a very small sample scenarios to illustrate the requirements for representing context. We will later come back to this scenario as a running example to demonstrate the context representation formalism.

Example 1 *Consider an ontology learning application that automatically extracts ontologies from a document corpus with abstracts in natural language. A fragment of the document with DOI* `http://dx.doi.org/10.1016/S0166-3615(03)00067-8` *in the corpus is:* A web-based system called Karlsruhe's virtual documentation (KaViDo) tool for collaborative research and development is presented. The objectives of KaViDo are to record and document development processes, to manage the competencies of distributed experts, to exchange user experiences and to assist product development. KaViDo is a browser-based tool. XML is used to exchange data between KaViDo and other applications.

An ontology learning tool (such as Text2Onto) might extract from the text that KaViDo is an instance of the classes tool and application. These facts we may formally want to represent in a domain ontology. Additionally, we may want to represent that Text2Onto was the agent producing these facts, and that these facts are afflicted with uncertainty (e.g. a degree of confidence).

In this example we observe a combination of two levels of information that need to be represented: The first type of statements directly concerns the application domain (the domain described in the document corpus, e.g. software tools). We refer to these statements as *domain-level* information. Additionally, we encounter facts that do not directly talk about the application domain, but that describe the domain-level information itself (e.g. facts about the source of information), i.e. context information about the facts. In the following, we discuss the limitations of current ontology languages w.r.t. the support of domain- and context information and derive requirements that need to be fulfilled for representing context information.

In OWL as the knowledge language, statements about the domain in Example 1 can be represented using the following ontology O_D.

(1) ClassAssertion(*a:Tool a:kavido*)

Note, however, that O_D represents the extracted information only partially, it does not capture the context information about source and confidence of the fact. In OWL, accommodating this context information would require modifying the domain ontology. Context and domain information would then be interpreted within a single model. Certain semantic interactions between these two levels of information would be possible, which lead to unintended consequences or even undecidability of reasoning [Mot07].

To avoid this, applications must manage domain- and context information explicitly, which is often cumbersome and error prone. It is therefore desirable to shift this burden to the ontology management infrastructure, which should provide the management of domain- and context information as a standardized service.

The representation framework should ensure a clear logical separation between the two types of information in two different ontologies. These two ontologies, however, should not be completely isolated, as this would make keeping all aspects of information in synchrony quite difficult. Ideally, the storage mechanism should allow us to physically group domain- and metalevel information, but interpret each one of them independently. Besides, interactions between these two levels of information should be supported in a controlled manner by a query language.

3.2 Overview of the Approach

We present a simple, but yet semantically sound framework for the representation of context information that can easily be integrated into current ontology management systems and reasoners. Our framework is based on the observation that the domain and context information have distinct universes of discourse.

Concretely, we propose two alternative mechanisms for storing context information in an OWL ontology in a way that does not affect the semantics of domain information, as shown in Figure 3.1. In the first alternative (1), context information is represented as annotations to the axioms of the domain ontology. I.e., domain and context information are managed in the same ontology, but as the annotations constitute non-logical information, the interpretation domains do not conflict. In the second alternative, domain information and context information are managed in two separate ontologies with independent first order models. The relationship between the domain and context information is established by providing a so-called *metaview* on the domain ontology within the context ontology. The metaview reifies the domain ontology and thus allows to make statements about the elements in the domain ontology.

Our framework depends on certain features of OWL 1.1[1]—an extension of OWL that is currently being standardized by W3C. We assume that the reader is familiar with OWL 1.1 concepts and its Functional-Style Syntax, which we use in the examples in the following.

3.3 Representing Context as Annotations

With this alternative, domain- and context information are stored in the same physical ontology using OWL 1.1 *annotations*. OWL 1.0 already allowed for annotations on ontology entities: concepts, properties, and individuals can have information that is akin to comments in programming languages. OWL 1.1 extends this idea to axioms as well. Thus, annotations in OWL 1.1 have the form (2) or (3), depending on whether the value of the annotation is an individual or a constant. Such annotations can be associated in OWL 1.1 with entities, axioms, and even ontologies.

(2) Annotation(*annotationURI individual*)

(3) Annotation(*annotationURI constant*)

[1] `http://www.webont.org/owl/1.1/`

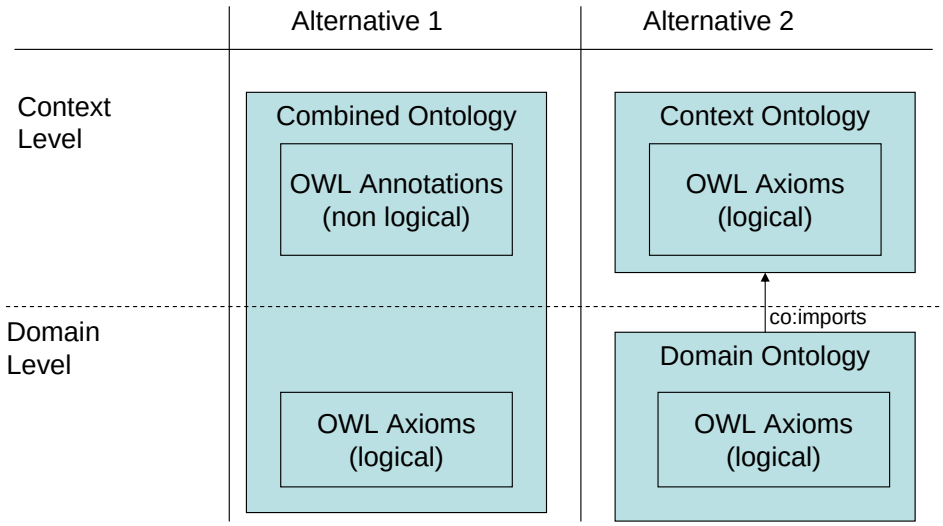

Figure 3.1: Alternatives for Representing Context

Thus, we represent statements from Example 1 using an ontology O_D that contains the following OWL axioms.

(4)
ClassAssertion(
Annotation(*a:source a:Text2Onto*)
Annotation(*a:confidence* "0.5")
a:Tool a:kavido)

In this way, domain- and context information is managed physically in one place. The annotations constitute non-logical information, meaning that their treatment is outside the (regular) semantics of OWL. They can thus be stored within the original ontology without causing undesired semantic effect to domain-level information.

3.4 Representing Context using Context Ontologies

In the second alternative, domain and context information are represented in two separate ontologies O_D and O_C. These ontologies is situated on a different meta-level: The domain ontology describes the domain of interest, the context ontology provides information about the facts in the domain ontology. The first-order models of O_D and O_C are independent.

O_D only represents the domain level information, for example:

(5)
ClassAssertion(
a:Tool a:kavido)

The ontology O_C consists of three main parts: (1) a reified representation of the axioms of O_D, (2) the logical representation of the context information for O_D, and (3) a description of the context (metalevel) domain.

Part (1) of O_C – the reified representation of O_D – we also call the *metaview* of O_D. For its representation, we created a metaontology O_{ODM} (ODM standing for Ontology Definition Metamodel) that captures the structure of OWL 1.1 ontologies.[2] This metaontology is directly based on the networked ontology model described in [HBP+07].

[2] http://owlodm.ontoware.org/OWL1.1

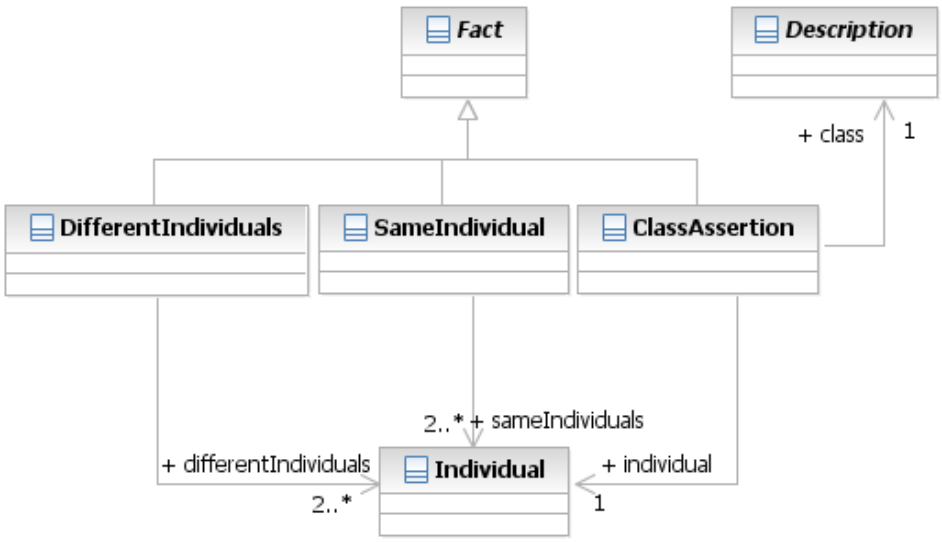

Figure 3.2: Facts represented in the OWL 1.1 Metaontology

Figure 3.2) shows a UML-based representation of a part of this metaontology, describing the facts in an ontology.

For example, a class assertion is described in O_{ODM} using the following axiom (*mo:* is the namespace prefix used in O_{ODM}); it states that a class assertion is a kind of fact and that it has exactly one class and one individual.

SubClassOf(*mo:ClassAssertion*
  ObjectIntersectionOf(*mo:Fact*
    ObjectExactCardinality(1 *mo:class mo:Description*)
    ObjectExactCardinality(1 *mo:individual mo:Individual*)))

In appendix A, we define a function μ, that defines the mapping from a domain-level O_D ontology into its metaview representation according to the metaontology.[3] Each axiom α of O_D is represented in O_C as a unique fresh individual x_α. Then, α is represented in O_C by following the structure defined by O_{ODM}. For example, assertion (5) is assigned an individual $x_{(5)}$, and it is represented with the following set of assertions in O_C.

(6) ClassAssertion(*mo:ClassAssertion* $x_{(5)}$)

(7) ObjectPropertyAssertion($x_{(5)}$ *mo:class a:Tool*)

(8) ObjectPropertyAssertion($x_{(5)}$ *mo:individual a:kavido*)

Part (2) of O_C represents the actual context information in the form of statements about the reified facts of the domain ontology. Thus, the context information is interpreted in O_C as domain level information. For example, the context information about the fact (5) is translated into the assertion (10).

(9) ObjectPropertyAssertion($x_{(5)}$ *a:source a:Text2Onto*)

(10) DataPropertyAssertion($x_{(5)}$ *a:confidence* "0.5")

Part (3) of O_C axiomatizes the context (metalevel) domain. For example, the statement that Text2Onto is a high quality source can be represented by axiom (11). This information can be quite complex and may involve advanced OWL 1.1 constructs; for example, (12) says that manually entered information comes from

[3] We provide a default implementation in KAON2. This implementation is able to generate the metaview on the domain ontology O_D on the fly, i.e. when interpreting O_C. As such the reified ontology does not need to be physically managed, but only needs to exist as a virtual ontology.

some (unknown) high-quality source.

(11) ClassAssertion(*a:Text2Onto a:HighQuality*)

SubClassOf(
(12) ObjectHasValue(*a:agent a:Manual*)
ObjectSomeValuesFrom(*a:source a:HighQuality*))

As the axiomatization of the context information is placed in a separate ontology O_C, the relationship between O_D and O_C needs to be established, This is done in the following way: The domain-level ontology O_D can contain an ontology annotation with a special predefined URI *co:import* and the value being equal to O_C. This instructs to interpret O_C as context information for O_D.

3.5 Comparison and Translation from Annotations to Context Ontologies

The two alternative representations presented above exhibit advantages and disadvantages that make them applicable for different use cases.

When representing context information as annotations, the context information constitutes non-logical information, i.e. it is disregarded by a reasoner. Instead one can think of the annotations as comments. If reasoning over the context information is not required, this simple representation is typically sufficient. A drawback is that the annotations can only have a simple structure (OWL 1.1 currently only supports annotations by entity and by constant, but not by more complex structures). The main advantage lies in the fact that domain- and context information can physically be managed in a single ontology.

The main advantage of representing context information in a separate context ontology is that context information can be queried and reasoned over using standard ontology reasoners, as the context information itself constitutes logical information. It is also possible to query domain information and context information in an integrated way. Further, more complex context structures can be represented (using all primitives available in the ontology language). The main disadvantage lies in the increased complexity and the fact that two separate ontologies for domain and context information need to be managed.

The decision for one of the alternatives is not irreversible: In appendix A, we define a function μ, that takes a domain-level O_D ontology containing axioms with context information embedded in annotations and translates it into another ontology $\mu(O)$, that corresponds to the context ontology O_C. The basic idea of this translation function is that the logical information in the domain ontology becomes reified in O_C (corresponding to part (1) of O_C), while the non-logical annotations (the context information) are turned into logical information (corresponding to part (2) of O_C). The inverse translation is only possible for simple context information (that can be translated into annotations by entities and constants), more complex structures would require complex annotations, which are currently not available in OWL1.1.

3.6 Summary

In this chapter, we have presented a proposal how represent context information in OWL ontologies. The approach is directly built on the networked ontology model and is compatible with the OWL 1.1 language, support for which is provided in the NeOn infrastructure.

In the approach, we distinguish two alternatives for managing the context information: In the first alternative, context information is represented as annotations to the elements of domain ontology. In the second alternative, context information itself is represented as logical information in an ontology separate to the domain ontology. The relationship with the elements of the domain ontology is established by reifying the domain ontology using a meta-ontology.

The representation formalism is generic in the sense that it can accommodate arbitrary forms of context languages. For a given context language one needs to define its vocabulary in the form of a context ontology. In the following chapter, we will provide context vocabularies for specific forms of context.

Chapter 4

Instantiation

In this chapter we instantiate our generic definition of context for three specific forms of context: *Provenance*, *Arguments* and *mapping*. Provenance is a form of context that is typically available for automatically generated ontologies – e.g. in ontology learning; Argumentation Structures are a form of context that is obtained in collaborative ontology engineering processes; Mappings are a form of context that is obtained by ontology mapping systems to achieve an interoperation between applications or data relying on different ontologies.

NeOn has in its core the ambitious scenario that ontologies are developed in the open environment in a distributed fashion. Moreover, it is not just the ontologies and meta-data that are distributed, but we also assume that they are built by distributed teams. In terms of the usages of context, this means that *supporting viewpoints and perspectives* will play a paramount role in NeOn. In the scenarios addressed by NeOn, information sources typically cannot be easily integrated without violating the overall consistency of the system. Thus *dealing with inconsistent information* will be another important usage of context, where information about the provenance of ontological structures, about various contexts and user profiles leads to the generation of local consistent views out of a globally inconsistent network of ontologies. Closely related is the problem of *dealing with uncertain and vague information*, which will play an important role in the NeOn scenarios, where ontologies are generated from a variety of sources that may be imprecise, vague and contextualized in the first place (e.g. natural language text) and where automated ontology learning algorithms introduce an additional dimension of uncertainty.

4.1 Provenance

4.1.1 Provenance Context

Provenance is a form of context that is typically available for ontologies. It shows how statements are organized among agents on the web and where the sources of the ontology elements are. In NeOn deliverable D2.3.1, two kinds of provenance are considered: *data provenance* and *provenance supporting collaborative ontologies*. Data provenance is usually related to the instance level of an ontology. It has been widely used in the area of database systems. The role of provenance in collaborative ontology engineering is to track the reasons why a change has occurred and to record the history of design process. In this section, we focus on data provenance.

In the following, we introduce a model for representing provenance. The main part of the model is shown in Figure 4.1. We associate with each of the ontology elements a confidence and a relevance value. The confidence value indicates how confident the system is about the correctness of an ontology element. Relevance value denotes the relevance of an ontology element with respect to a particular domain given by a source.

In addition to confidence and relevance values each ontology element is associated with some provenance elements that can be used to generate formal or informal explanations for particular results. The source of a provenance element is an InformationObject. A provenance element can be also provided by an agent. We also associate with each of the provenance elements a confidence and a relevance value. The confidence

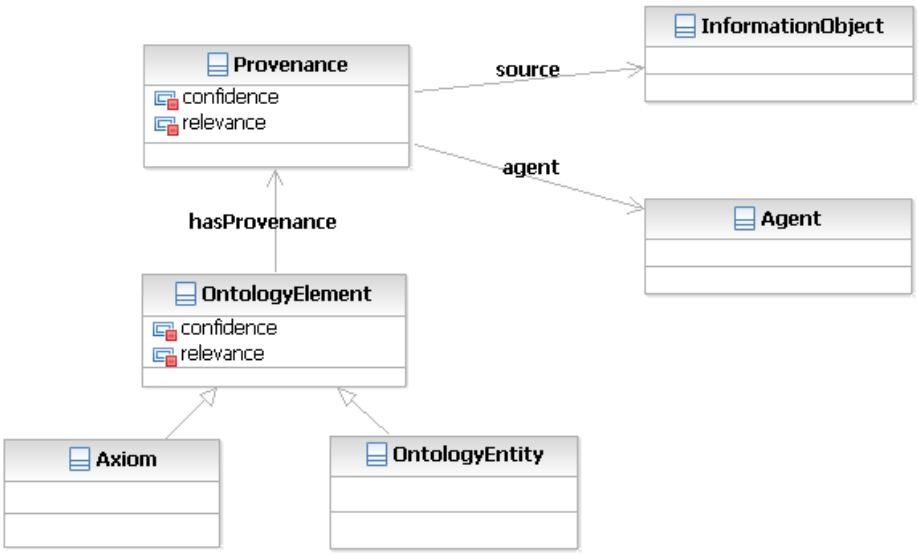

Figure 4.1: Provenance Context

(resp. relevance) value of an ontology element is computed by aggregating confidence values of those provenance elements associated with the ontology element. Both confidence and relevance values can be considered as context annotations. Typical examples for sources of provenance elements which may lead to the creation of a `subclass-of` relation, for instance, include Hearst patterns [Hea92] and hyponymy relationships in WordNet [Fel98].

The model given by Figure 4.1 is general and can be grounded for different purpose. Figure 4.2 gives one grounding. An InformationObject has three subclasses: WeightedReference, Term and Document. We associated with each of the WeightedReference elements a weight which refers to an InformationObject. Let us illustrate this model by a grounded ontology in Example 2 that can be obtained by the ontology population module given in NeOn deliverable D3.2.1. In this ontology, the concept X is associated a set of pairs (Term, Weight), i.e., pair(term1,0.3).

Example 2 *Consider we want to ground an ontology by relating the ontology elements with a set of pairs (Term, Weight), e.g. concept(X, [pair(term1, 0.3), pair(term2, 0.9), ...]).*

In the context ontology, this information would be represented as follows (in OWL Functional Syntax):

```
ClassAssertion(odm:OWLClass X)
```

Figure 4.2: Provenance: Grounding Ontology Elements

```
ObjectPropertyAssertion(hasProvenance X p1)

ClassAssertion(Provenance p1)
ObjectPropertyAssertion(source p1 r1)
ObjectPropertyAssertion(source p1 r2)

ClassAssertion(WeightedReference r1)
ObjectPropertyAssertion(refersTo r1 term1)
DataPropertyAssertion(weight r1 "0.3")

ClassAssertion(WeightedReference r2)
ObjectPropertyAssertion(refersTo r2 term2)
DataPropertyAssertion(weight r1 "0.9")
```

Example 3 *Based on the grounding with term vectors as shown in Example 2, one may also want to represent the (cosine) distance between two concepts as contextual information, e.g. distance(concept(X),concept(Y),0.8). This could be done in the following way:*

```
ClassAssertion(OWLClass X)
ClassAssertion(OWLClass Y)


ClassAssertion(Distance d_XY)
ObjectPropertyAssertion(entity1 d_XY X))
ObjectPropertyAssertion(entity2 d_XY Y)
DataPropertyAssertion(distance d_XY "0.8")
```

4.1.2 Usage of Provenance Context

In the following, we consider how provenance context can be exploited to deal with inconsistency and uncertainty.

In many cases, the information derived from diverse sources leads to inconsistencies. This is especially the case if information is derived using automatic knowledge acquisition tools such as wrappers [FK00] or

information extraction systems (e.g. [Cir01], [BCRS06]) or tools for automatic or semi-automatic ontology learning such as OntoLT [BOS03], OntoLearn [NVCN04], ASIUM [FN98], OntoGen [FGM07] or Text2Onto [CV05] that aim at the semi- or even fully automatic extraction of ontologies from sources of textual data. Common to all of them is the need for handling the uncertainty which is inherent in any kind of knowledge acquisition process. Moreover, ontology-based applications which rely on learned ontologies have to face the challenge of reasoning with large amounts of imperfect information resulting from automatic ontology generation systems.

Different causes for the imperfection of information can be identified. According to [AM97] imperfection can be due to *imprecision*, *inconsistency* or *uncertainty*. Imprecision and inconsistency are properties of the information itself - either more than one world (in the case of ambiguous, vague or approximate information) or no world (if contradictory conclusions can be derived from the information) is compatible with the given information. Uncertainty means that an agent, i.e. a computer or a human, has only partial knowledge about the truth value of a given piece of information. One can distinguish between objective and subjective uncertainty. Whereas objective uncertainty relates to randomness referring to the propensity or disposition of something to be true, subjective uncertainty depends on an agent's opinion about the truth value of information. In particular, the agent can consider information as unreliable or irrelevant.

Depending on the type of imperfection, different approaches for interpreting the information may be adequate. For each of them, the provenance of information can be exploited, as we have outlined in the deliverable D3.1.1 [HHR$^+$06].

There are mainly two classes of languages for dealing with uncertainty: probabilistic logic and possibilistic logic. Dealing with probabilistic uncertainty in description logics has been recognized as an important problem since 1994. Many approaches have been proposed to extend description logics with probabilistic reasoning [Jae94, Hei94, GL02, DS05, NF04, DP04, KLP97]. These approaches can be classified according to ontology languages, the supported forms of probabilistic knowledge and the underlying probabilistic reasoning formalism. In probabilistic extensions of description logics, a probability value (or an interval) is often attached to a *conditional constraints* of the form $(D|C)$, where C and D are concepts. By contrast, there is relatively few work on combining possibilistic logic and description logic. Possibilistic logic [DLP94] or possibility theory offers a convenient tool for handling uncertain or prioritized formulas and coping with inconsistency. It is very powerful to represent partial or incomplete knowledge [BLP04]. There are two different kinds of possibility theory: one is qualitative and the other is quantitative. Qualitative possibility theory is closely related to default theories and belief revision [DP91, BDP92] while quantitative possibility can be related to probability theory and can be viewed as a special case of belief function [DP98]. One of the major problems with the quantitative possibility theory is that the weights attached to formulas are usually hard to obtain. When numerical information is not available, we often use qualitative possibility theory. In this case, a possibilistic knowledge base can be viewed as a stratified knowledge base, i.e. knowledge bases in which all pieces of information are assigned a rank. The confidence values attached to axioms in possibilistic description logics are often used to represent priority levels of the axioms. A challenging problem here is to extract ranking information from an inconsistent or incoherent ontology.

Fusing mutually inconsistent knowledge extracted from heterogeneous sources in essence requires (i) an algorithm to pinpoint down where the inconsistencies arise, (ii) a procedure to resolve inconsistencies by removing axioms leading to these inconsistencies, as well as (iii) a representation of the provenance of axioms as context information on the basis of which to decide which axioms should be removed.

For the knowledge fusion scenario, we intend to build on the approach of [HV05] to find *minimally inconsistent axioms sets* within a given ontology. The idea behind minimally inconsistent axiom sets is that the removal of one axiom in each set will lead to consistency. The decision which axiom to remove can then indeed be guided by the provenance context as described above.

Axiom t	Confidence
$Architecture \sqsubseteq \neg Tool$	0.10
$Methodology \sqsubseteq \neg Tool$	0.10
$Tool \sqsubseteq Implementation$	0.40
$Application(kavido)$	0.46
$Tool(kavido)$	0.46
$Tool(amilcare)$	1.0
$Tool \sqsubseteq \neg Application$	0.3

Table 4.1: Ontology learning example

4.1.3 Ontology learning example

In this subsection, we use an example from [HV05] to illustrate how confidence values attached to axioms of an ontology can be used to deal with imperfect information. In this example, the ontology is obtained by ontology learning tool Text2Onto reported in NeOn deliverable D3.8.1.

Let us consider the ontology in the table 4.1. The example exhibits two forms of imperfection: First, the elements of the learned ontology are uncertain, as indicated by the confidence values that are attached to the axioms in the ontology. Second, the obtained knowledge is inconsistent: Here $KaViDo$ was identified to be both an instance of $Application$ and a $Tool$, however, $Application$ and $Tool$ were learned to be disjoint concepts, so the ontology is inconsistent.

We may consider two ways to deal with inconsistency in the ontology: we can either delete some axioms to restore inconsistency or tolerate the inconsistency and apply possibilistic logic approach.

To resolve inconsistency, we first need to find the minimally inconsistent sub-ontologies of the inconsistent ontology based on the algorithm. There is only one minimally inconsistent sub-ontology which contains the following axioms: $Tool \sqsubseteq \neg Application$, $Application(kavido)$, $Tool(kavido)$. Taking the confidence values of the axioms into account, we can resolve the inconsistency by removing the disjointness axiom whose confidence value is the lowest. In this case, removing the axiom $Tool \sqsubseteq \neg Application$ would yield a consistent ontology.

We next illustrate how to reason with the inconsistent ontology using possibilistic logic. In this case, the confidence values attached to axioms are explained as certainty degrees of the axioms. We first need to find the *inconsistency degree* of this ontology, which is the maximal weight of axioms such that all the axioms whose weights are greater than or equal to it are inconsistent. An axiom can be inferred from the ontology using possibilistic inference if and only if it can be inferred from axioms whose weights are greater than the inconsistency degree. For the ontology in this example, its inconsistency degree is 0.3. So, we can infer $Tool(kavido)$ whose weight is 0.46, which is greater than 0.3. Furthermore, we can also infer $Implementation(amilcare)$ with confidence degree 0.40 using possibilistic inference.

4.1.4 Instantiation of the Generic Definition

Let K be a knowledge base in an ontology language L. The context information is the confidence and relevance values attached to formulae in the knowledge base. In the case of probabilistic logic, the context language is the conditional probabilistic terminology for probabilistic logic or conditional probability table for Bayesian networks. The semantics of the context language is a knowledge base which consists of K and probabilistic terminologies. In the case of possibilistic logic, the context information can be used to obtain the weights or priority levels attached to formulae in the knowledge base. That is, $S(K, C)$ is a weighted knowledge base or a prioritized knowledge base.

We give more detailed explanations on how to use possibilistic OWL to represent and reason with provenance context. We suppose that ontologies consist of a TBox and an ABox in the description logic $\mathcal{SHOIQ}(D+)$. Given an ontology $O = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, a possibilistic ontology corresponding

to O is a weighted ontology $O_P = \langle \mathcal{T}_P, \mathcal{A}_P \rangle$ such that $\mathcal{T}_P = \{(\phi_i, \alpha_i) : \phi \in \mathcal{T}\}$ and $\mathcal{A}_P = \{(\psi_j, \beta_j) : \psi_j \in \mathcal{A}\}$, where $\alpha_i, \beta_j \in [0, 1]$. To define the consequence relation of a possibilistic ontology, we need introduce some notions.

Given a possibilistic ontology $O_P = \langle \mathcal{T}_P, \mathcal{A}_P \rangle$ and $\alpha \in (0, 1]$, the α-cut of $\mathcal{T}_P$ is $\mathcal{T}_{P,\geq\alpha} = \{\phi \in \mathcal{T} | (\phi, \beta) \in \mathcal{T}_P \; and \; \beta \geq \alpha\}$ (the α-cut of $\mathcal{A}$, denoted as $\mathcal{A}_{P,\geq\alpha}$, can be defined similarly). The α-cut of O_P is $O_{P,\geq\alpha} = (\mathcal{T}_{P,\geq\alpha}, \mathcal{A}_{P,\geq\alpha})$. The *inconsistency degree* of O_P, denoted $Inc(O_P)$, is defined as $Inc(O_P) = max\{\alpha : O_{P,\geq\alpha} \; is \; inconsistent\}$.

The consequence relation of possibilistic OWL DL is defined as follows: An axiom ϕ (which is either a TBox axiom or an ABox assertion) can be inferred from a possibilistic ontology O_P, written $O_P \models_P \phi$, if $O_{P,>Inc(O_P)} \models \phi$, where $O_{P,>Inc(O_P)} = O_{P,\geq Inc(O_P)} \setminus \{\phi \in O : (\phi, \alpha) \in O_P \; and \; \alpha = Inc(O_P)\}$.

4.2 Argumentation

Argumentation frameworks usually provide information that can be useful to provide information about the context of ontology elements.

The two classical approaches on argumentation theory [Tou58, POT70] provide general schemas for representing argumentative processes. However, argumentation is a large field in continuous evolution and proposed frameworks vary from informal to rigorous formal ones. One acknowledged problem of these theories is the fact that it is difficult to find the appropriate level of detail with which to represent arguments. For some more details about argumentation theories see D2.1.1. In the case of ontology based argumentation frameworks, DILIGENT [TPSS05] proposes an argumentation ontology that can be used to formally represent the arguments exchanged by ontology engineers in ontology building processes. Some initial experiments have been conducted. Another framework currently under development under the Neon Project is C-ODO [CGL$^+$06].

These frameworks usually provide important information that can be used in context representation and moreover that can be used in inference processes. In the following paragraphs we are taking as starting point the DILIGENT argumentation framework [TPSS05, Tem06], which is the core of the Cicero framework given in NeOn deliverable D2.3.1. The DILIGENT argumentation ontology adapts, for ontology engineering purposes, the IBIS methodology, which proposes one model to formally structure arguments. The DILIGENT argumentation framework consists of two building blocks; a process and an argumentation ontology. In the argumentation process five main activities take place: choose moderator, choose decision procedure, specify issues, provide arguments and ideas, decide on issues and ideas. The argumentation ontology formalizes the arguments exchanged during ontology engineering discussions.

The DILIGENT Argumentation Ontology is visualized in Figure 4.3. It formalizes the arguments exchanged during ontology engineering discussions. The main concepts in this ontology are issues, solution proposals and arguments. An *issue* introduces a new requirement or topic in a discussion from a conceptual point of view. Issues may refer to particular ontology elements. They are used to discuss problems in the definition of ontology elements without yet taking into account how the problems should be resolved and implemented in the ontology. *Solution proposals* are put forward as ideas to *address issues* and refer to their formalization in the ontology, for instance as a class, instance, relation or axiom. Typically, a solution proposal encompasses one or more ontology changes that may affect the definition of ontology elements. New requirements or topics are introduced as issues, which can be extended or refined by *generalizing* or *specializing* an issue. When the experts start discussing how a given issue, a domain concept, can be represented in the ontology, they discuss in terms of solution proposals, how domain knowledge can or should be formalized. Accepted solution proposals trigger concrete ontology change operations. *Arguments* can be exchanged on particular solution proposals, either supporting an idea or objecting (counter-argument).

The concepts an ontology represents should be consensual, this requires some consensus building discussions. In DILIGENT processes, concepts are only added to the ontology if they can be agreed upon, that is after some arguments have been exchanged, positions by different actors have been issued on them and

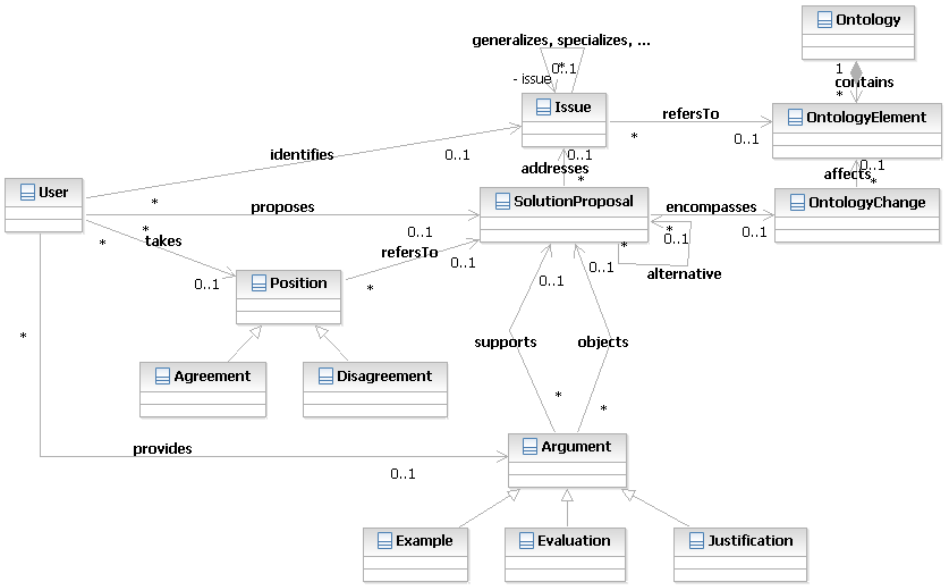

Figure 4.3: The major concepts of the argumentation ontology and their relations

some decisions have been made. DILIGENT proposes examples, evaluations and justifications as particularly useful argument types. Those involved in discussions can state *positions*. They clarify the position on one of particular solution proposals under discussion. Possible positions are *agreement* and *disagreement*. Once enough arguments have been provided and positions have been stated on them, decisions can be made. In general, positions lead to decisions. Decisions are taken on issues. A decision has a status that can vary from under-discussion, postponed, discarded and agreed. A decision records not only the issue on which it was taken, but also both the positions issued when final with-votes (several positions) were cast and the line of reasoning (a sequence of arguments) underlying the decision on that issue.

The argumentation structures can be exploited as context information in various ways in the ontology reasoning. In the following we will illustrate how this can be done for the task of diagnosis and repair. Diagnosis and repair of ontologies is an important aspect of ontology engineering. This is especially the case in collaborative engineering environments, where there may be conflicts and disagreements about the meaning and definition of concepts among the ontology engineers.

Reasoning agents can be useful in different steps of the argumentation process: Indeed, in every phase they can act in place of a user, i.e. they can *identify issues*, they can *propose solutions*, they can *take positions* and they can *provide arguments*. Identifying *issues* corresponds to the identification of a logical contradiction in the ontology. To resolve the logical contradiction, it is important to pinpoint the erroneous class definitions in the ontology which are responsible for the contradiction. (Please refer to [QHJ07] for an overview of different approaches for this task.) When repairing an ontology, we usually have different alternatives to resolve the incoherence, i.e. the diagnosis algorithms may generate a number of different *solution proposals*. Some context information such as ranking information is often needed to select the best solution [KPSG06]. The

class definitions	PhDStudents⊑Students	PhDStudent⊑Empolyees	Student⊓Employees⊑ ⊥
rank	0	1	2

Table 4.2: Ranking on class definitions

idea here is to use the argumentation structures as context information to define useful rankings for the different solution proposals. Based on the ranking, the reasoning agent would define a *position* on a solution proposal, i.e. recommending one (or multiple) solution proposals to implement. The final decision can then either be done automatically (by accepting the recommendation of the reasoning agent) or it can be left to the user. If the decision is left to the user, the individual arguments about the solution proposal can be presented to the user (based on the argumentation ontology) in order to support the decision process.

4.2.1 The University example

Let us suppose we want to model the University domain with the purpose of inferring about the properties of the several members of the University, find inconsistencies and be able to reason in several contexts. In a DILIGENT ontology engineering process participants would start by proposing issues and after enough arguments had been provided and that participants had reached an agreement as to what should be represented in the ontology, the arguments would be attached to ontology elements. We will be using the real data from the discussions that took place in one of the DILIGENT case studies, partially reported in [PST04]. We here assume that the ontology targeted by Figure 4.4 (see page 32) contains concepts such as organization, and in particular university, persons, employees, Professors, students, PhD students, study programs. All these concepts are all related since university employees work at universities, both Professors and PhD students are university employees, students are enrolled in study programs and these are offered by universities. In the figure, we use the circle symbol to denote the disjointness of two concepts Students and (Uni)Employees.

This example contains a contradiction: PhDstudents are both subclass of Students (PhDstudents⊑Students) and University Employees (PhDstudents⊑Employees), and these classes are disjoint (Students⊓Employees⊑⊥). To resolve the contradiction, we can get an ordering relation on the axioms which are involved in the conflict based on argumentation information. Then we can simply delete the axiom which has least priority.

Using the argumentation ontology, we can represent the following arguments:

Arguments for PhD students being students could be:

Pro - enrolled in student programm

Cons - pay taxes

Arguments for PhD students being Employees

Pro - Work on projects; Have a salary

Cons - have a scholarship; do not have a contract; do not pay taxes; (UK)

Arguments for students disjoint from employees

Pro - not payed (China); do not work; enrolled in study program

Cons - people that work and study in the evening

To resolve the incoherence in the University ontology, we have at least the following three solutions: (1) delete the class definition PhDStudents is a subclass of Students, or (2) delete the class definition PhDStudents is a subclass of Empolyees, or (3) delete the class definition “Students disjoint from Employees". However, without extra information, we do not know which class definition should be deleted. In this case, we can turn to argumentation ontology to get some ranking information on the class definitions. In the argumentation ontology, the difference between numbers of argument support and against “PhD students being students" is 0, the difference between numbers of argument support and against “PhD students being employees" is 1, and the difference between numbers of argument support and against “students disjoint from employees" is 2 (see Table for details). Therefore, we can conclude that the class definition that “PhD students being

students" is has the least priority. So we can simply delete this class definition to restore coherence.

4.2.2 Instantiation of the Generic Definition

Let K be a knowledge base in an ontology language L. The context information C is then arguments which are provided by actors and is stored in an argumentation ontology. The context language L_c is the ontology language that is used to represent the argumentation ontology. The semantics of K in the context C is then defined as follows: $S(K, C)$ is the set of axioms that can be inferred from K using information provided by C. For example, if we use C to give a rank on axioms in K and get a new ontology K' where all axioms in K' are assigned a rank, and we apply possiblistic logic to infer new conclusions, then $S(K, C) = \{\phi : K' \models_P \phi\}$. It is clear that the context semantics is conservative and idempotent. According to the University example, it is clear that the context semantics is not extensive because the axiom “PhD students being students" is not included in the final result. It is neither knowledge-monotone nor context-monotone. For knowledge-monotone, if we add an axiom $PhDStudents(John)$ to state that John is a PhD student and an argument for it and an argument against it. Then we may delete the axiom $PhDStudents(John)$ and keep the class definition that “PhD students being students". So knowledge-monotone is violated. For context-monotone, suppose we have more arguments support the class definition “PhD students being students", for example, PhD students need to attend lectures and PhD students have a student card, then we may delete the class definition “PhD students being employees" to restore consistency.

4.3 Mapping

When people are modeling the same domain, they mostly produce different results, even when then use the same language. Mappings have to be defined between these ontologies to achieve an interoperation between applications or data relying on these ontologies. In NeOn deliverable D3.1.1, we have discussed how mappings between ontologies supply context. For example, in C-OWL (Context OWL) [BGvH$^+$03], multiple ontologies are treated as separate entities, between which information interchange is realized only by explicit mappings (so-called *bridge rules*). This chapter provides an extension for the metamodel for OWL and SWRL to give additional support for mappings between heterogeneous ontologies.

Section 4.3.1 introduces the metamodel extension for OWL ontology mappings. While introducing the various mapping aspects[1], we discuss their representation in the metamodel. Accompanying UML diagrams document the understanding of the metamodel.[2]

4.3.1 A Common Metamodel for OWL Ontology Mappings

This section presents the common metamodel extension for OWL ontology mappings in two subsections: The first subsection presents mappings, after which the second subsection presents queries.

Mappings

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

- A mapping is a set of mapping assertions that consist of a semantic relation between mappable elements in different ontologies. Figure 4.5 demonstrates how this structure is represented in the meta-

[1] Remember, however, that the OWL ontology mapping languages and their general aspects, are not part of our contribution.
[2] In doing so, meta-classes that are colored or carry a little icon again denote elements from the metamodel for OWL or SWRL.

model by the five metaclasses *Mapping*, *MappingAssertion*, *Ontology*, *SemanticRelation* and *MappableElement* and their associations.

- Mappings are first-class objects that exist independent of the ontologies. Mappings are directed, and there can be more than one mapping between two ontologies. The direction of a mapping is defined through the associations *sourceOntology* and *targetOntology* of the metaclass *Mapping*, as the mapping is defined from the source to the target ontology. The cardinalities on both associations denote that to each *Mapping* instantiation, there is exactly one *Ontology* connected as source and one as target.

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

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

The question of what is preserved by a mapping is tightly connected to the hidden assumptions made by different mapping formalisms. A number of important assumptions that influence this aspect have been identified and formalized in [SSW05]. A first basic distinction concerns the relationship between the sets of objects (domains) described by the mapped ontologies. Generally, we can distinguish between a global domain and local domain assumption:

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

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

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

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

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

A *MappingAssertion* may optionally be attributed with a *confidence* value that indicates the degree to that the assertion can be trusted or considered reliable.

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

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

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

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

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

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

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

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

The set of semantic relations can be extended by additional types of relations if needed. For example, the SKOS vocabulary[3] defines as possible relations: `skos:semanticRelation`, `skos:broader`, `skos:narrower`, `skos:related`, `skos:broaderTransitive`, `skos:narrowerTransitive`. However, the semantics of these relations is not formally defined. We therefore do not include them as predefined relations in our model.

As mappable elements, the metamodel contains the class *OWLEntity* that represents an arbitrary part of an ontology specification. While this already covers many of the existing mapping approaches, there are

[3] `http://www.w3.org/TR/skos-reference/`

a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between (conjunctive) queries to connect models, which leads to a considerably increased expressiveness. These queries are represented by the metaclass *OntologyQuery*. Note that the metamodel in principle supports all semantic relations for all mappable elements.

Queries

A mapping assertion can take a query as mappable element. Figure 4.6 demonstrates the class *Query* that reuses constructs from the SWRL metamodel.

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

4.3.2 Concrete Syntax using DL-safe Mappings

The mappings as described in the mapping metamodel can be represented in a separate context ontology, as we have demonstrated it already for the argumentation and provenance ontology in the previous sections.

However, in the case of mappings as contextual information, it may also be desirable to interpret the mappings in the same interpretation domain as the domain ontologies themselves. For this, we can define specific groundings of the mappings in specific mapping languages. In [HBP+07] we have discussed several of such groundings, including for C-OWL and DL-safe mappings.

We here provide a small example of how a grounding of the mappings would be realized in DL-safe mappings [HM05] where the mappings can be discovered by the Alignment Server described in D3.3.1. Simply speaking, DL-safe mappings relies on OWL extended with the DL-safe subset of SWRL for the representation of the mappings. Many mapping patterns can indeed be represented in OWL directly. For example, a mapping assertion with an OWLClass as source and target element is represented as an *EquivalentClasses* axiom.

Example 4 *Suppose we want to express a mapping assertion that specifies equivalence as semantic relation between two classes X and Y with a confidence degree of 0.8.*

The mapping would be expressed as a DL-safe mapping as follows:

(1) EquivalentClasses*(* Annotation*(weight "0.8")* *X Y)*

4.3.3 Instantiation of the Generic Definition

We consider how to use C-OWL to instantiate mapping as a type of context against the generic definition. This can be done by two steps. We first use the weights attached to the mapping assertions to remove erroneous mapping assertions. We then apply the C-OWL semantics on the revised ontologies. For the first step, we can apply the debugging and repairing approach given in [MST07]. For the latter step, we consider a set of (local) ontologies O_i where $i \in I$. Choosing therefrom one local ontology O_k to focus on, let L_k be the local language of O_k, i.e. the set of statements only referring to concepts, individuals, and roles from O_k. Furthermore, let L be the language built up from all concepts, individuals, and roles from all ontologies of the context space. So obviously $L_k \subseteq L$. We use L_k as knowledge language and L as context language. The semantic of O_k is then the set $S(O_k) \subseteq L$ of the consequences we can derive when simply discarding

all "non-local" information, i.e. that about the other O_j with $k \neq j$ and all bridge rules $\{M_{ij}\}_{i,j \in \mathcal{I}}$. All other ontologies together with the bridge rules constitute the context C for O_k. We are aware, that our notion of context does not coincide with the use of the term in C-OWL, since in the C-OWL terminology, every O_k is conceived as a context on its own. However, we argue that our terminology captures the intention more precisely, since e.g. the information present in the mappings would in C-OWL terms be considered as outside of any context.

Obviously, O_k's "global theory" $S'(O_k, C)$ – all derivable consequences taking into account the distributed information of the considered context space – in general deviates from $S(O_K)$.

Characterizing this kind of semantic, we find that it is obviously conservative, extensive, knowledge-monotone, and context-monotone.

4.4 Summary

In this chapter, we considered three types of context which are of particular interest to NeOn project: provenance, argumentation and mapping. For each of these types of context, we gave an example to show how it can be represented by the syntax given in Chapter 3. The semantics of the context language was also discussed. By doing this, we related these three types of context to the general definition of context given in Chapter 2.

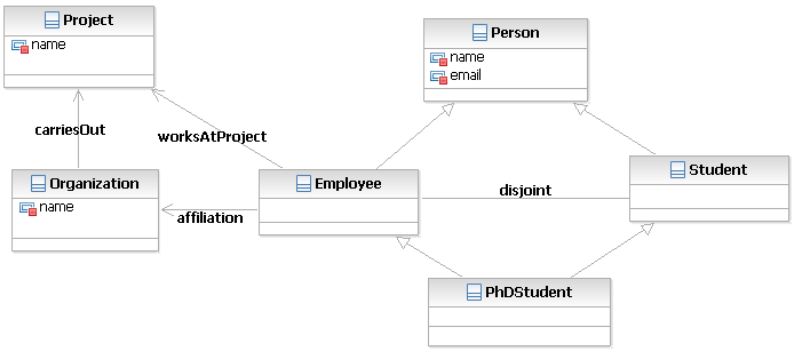

Figure 4.4: University ontology

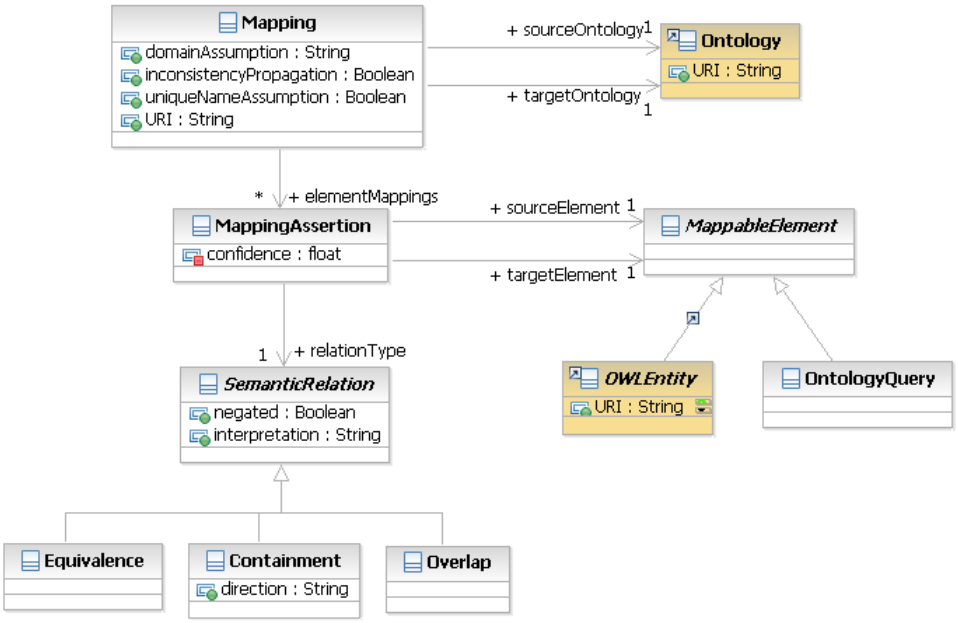

Figure 4.5: OWL mapping metamodel: mappings

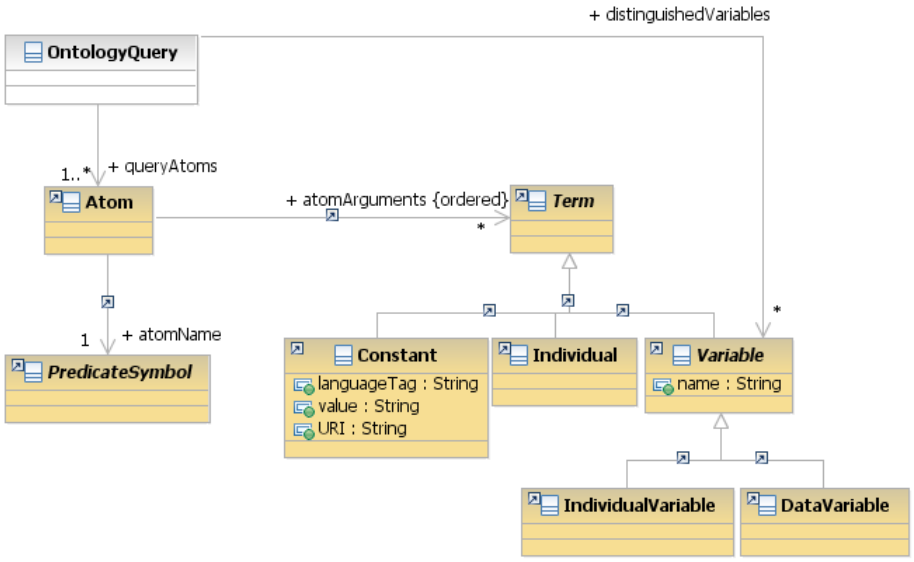

Figure 4.6: OWL mapping metamodel: queries

Chapter 5

Conclusion

5.1 Summary

In this deliverable, we defined the NeOn formalism for context representation. We first recalled the generic definition of context given in NeOn deliverable D3.1.1. We then presented a proposal how to represent context information in OWL ontologies. The representation formalism is generic in the sense that it can accommodate arbitrary forms of context languages. For example, it is used to represent the mappings as described in the mapping metamodel. For a given context language one needs to define its vocabulary in the form of a context ontology. So we provided context vocabularies for specific forms of context. We instantiated our generic definition of context for three specific forms of context and illustrated how they can be represented by the context ontologies.

5.2 Roadmap

We have developed context representation formalism which will serve as the foundation for representing context in the NeOn project. There are a couple of issues that are worthwhile for further investigation. First, we use OWL as an language to represent both ontology and context. There is some interest in NeOn project where rule languages like F-logic and RIF can be used. We may consider exploring how to use RIF to represent rules and context in the future. Second, we will combine the argumentation support tool developed in WP2 and debugging tool developed in WP1 so that we can use context information to deal with inconsistent information in ontologies. Third, in WP1, we have discussed how to apply context information to deal with inconsistency and provide evaluation of our approaches. However, we have not considered mappings as a context information for the purpose of debugging. As a future direction, we will incorporate mapping and discuss how to deal with inconsistency in networked ontologies.

Appendix A

Translation of an Ontology to its Metaview

This appendix contains the formal specification of the transformation μ, which takes an OWL 1.1 ontology $\mathcal{O}$ and generates its metaview $\mu(\mathcal{O})$. Given an ontology $\mathcal{O}$, we define the ontology $\mu(\mathcal{O})$ as follows:

$$\begin{array}{l}\mathsf{Ontology}(<mo\text{:}\textit{the ontology URI of } \mathcal{O}> \\ \quad \mathsf{Import}(http://owlodm.ontoware.org/OWL1.1) \\ \\ \quad \mu(\alpha) \textit{ for each ontology annotation } \alpha \textit{ in } \mathcal{O} \\ \quad \mu(\alpha) \textit{ for each axiom } \alpha \textit{ in } \mathcal{O} \\ )\end{array} \tag{A.1}$$

The ontology URI of $\mu(\mathcal{O})$ is obtained by prepending the ontology URI of $\mathcal{O}$ with mo. The ontology $\mu(\mathcal{O})$ imports the meta ontology of OWL 1.1 with the URI $http://owlodm.ontoware.org/OWL1.1$ ontology, which defines OWL classes and properties for describing the syntactic structure of $\mathcal{O}$.

The result of an application of the operator μ to ontology annotations is given in Table A.2. The result of an application of μ to axioms is defined as

$$\mu(\alpha) = \Lambda(\alpha) \cup \Xi(\alpha) \tag{A.2}$$

where the functions Λ and Ξ map axioms, class descriptions, data ranges, object property, data property expressions and lists in $\mathcal{O}$ to sets of ABox assertions, as defined in Tables A.1–A.14. These tables use the following abbreviations:

- `CLA` for ClassAssertion,
- `DPA` for DataPropertyAssertion, and
- `OPA` for ObjectPropertyAssertion.

The ontology $\mathcal{O}$ contains named and unnamed objects. Classes, properties, datatypes, and individuals are examples of named objects; each named object α is represented in $\mu(\mathcal{O})$ by an individual whose URI is equal to the URI of α; we denote this individual as x_α. In contrast, axioms, class descriptions, and property expressions are examples of unnamed objects. For each unnamed object α, the transformation μ generates a new individual that represents α in $\mu(\mathcal{O})$. To make the notation uniform, we denote this new individual also as x_α. We do not specify how to associate x_α with α; however, we assume that each implementation provides some way of obtaining x_α from α and vice versa.

OWL 1.1 provides for complex property inclusion axioms that involve a chain of object properties. Similarly, the restrictions on data properties can involve more than one property. In order to preserve the order of the properties in these elements in the axiom metaview, we treat this chain as a list of objects. We denote a list of objects as $L[o_1 \ \ldots \ o_n]$. Lists are unnamed objects. Hence, for a list α, we assume that there is a new

individual x_α that uniquely corresponds to α; furthermore, $x_{L[\,]} = mo{:}null$. Nonempty lists are translated into ABox assertions as shown in Table A.1.

Finally, OWL 1.1 allows each axiom to contain annotations. For all axiom types, their annotations are translated into assertions in the same way. If an axiom α is annotated with a URI ap and a constant value $\mathbf{ct}$, then $\Lambda(\alpha)$ contains the assertion (A.3). Similarly, if α is annotated with a URI ap and an individual in, then $\Lambda(\alpha)$ contains the assertion (A.4).

$$\mathsf{DataPropertyAssertion}(ap\ x_\alpha\ \mathbf{ct}) \tag{A.3}$$

$$\mathsf{ObjectPropertyAssertion}(ap\ x_\alpha\ in) \tag{A.4}$$

Table A.1: Translation of Lists

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
$L[e_1\ e_2\ \ldots\ e_n]$	$\texttt{OPA}(mo{:}first\ x_{L[e_1\ e_2\ \ldots\ e_n]}\ e_1)$ $\texttt{OPA}(mo{:}rest\ x_{L[e_1\ e_2\ \ldots\ e_n]}\ x_{L[e_2\ \ldots\ e_n]})$	$\mu(e_1) \cup \mu(L[e_2\ \ldots\ e_n])$

Table A.2: Translation of Ontology Annotations

α	$\mu(\alpha)$
AnnotationByConstant$(ap\ \mathbf{ct})$	$\texttt{DPA}(ap\ o\ \mathbf{ct})$
AnnotationByEntity$(ap\ in)$	$\texttt{DPA}(ap\ o\ in)$
Note: o is the URI of the ontology being translated.	

Table A.3: Translation of Entity Annotations

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
EntityAnnotation$(EType(e)\ ann_1\ \ldots\ ann_n)$	$\texttt{CLA}(x_\alpha\ mo{:}EntityAnnotation)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$ $\texttt{DPA}(mo{:}entityAnnotation\ x_\alpha\ x_{ann_1})$ $\ldots$ $\texttt{DPA}(mo{:}entityAnnotation\ x_\alpha\ x_{ann_n})$	$\bigcup_{i=1}^{n} \mu(ann_i)$
AnnotationByEntity$(ap\ in)$	$\texttt{OPA}(ap\ x_\alpha\ in)$	$\emptyset$
AnnotationByConstant$(ap\ \mathbf{ct})$	$\texttt{DPA}(ap\ x_\alpha\ \mathbf{ct})$	$\emptyset$
Note: $EType$ is one of OWLClass, Datatype, ObjectProperty, DataProperty, or Individual.		

Table A.4: Translation of Entities

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
OWLClass(cl)	$\texttt{CLA}(cl\ mo{:}OWLClass)$	$\emptyset$
Datatype(dt)	$\texttt{CLA}(dt\ mo{:}DataType)$	$\emptyset$
Individual(in)	$\texttt{CLA}(op\ mo{:}Individual)$	$\emptyset$
ObjectProperty(op)	$\texttt{CLA}(op\ mo{:}ObjectProperty)$	$\emptyset$
DataProperty(dp)	$\texttt{CLA}(op\ mo{:}DataProperty)$	$\emptyset$

Table A.5: Translation of Entity Declarations

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
Declaration(OWLClass (e))	$\texttt{CLA}(x_\alpha\ mo{:}OWLClassDeclaration)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$	$\mu(e)$
Declaration(Datatype (e))	$\texttt{CLA}(x_\alpha\ mo{:}DatatypeDeclaration)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$	$\mu(e)$
Declaration(ObjectProperty (e))	$\texttt{CLA}(x_\alpha\ mo{:}ObjectPropertyDeclaration)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$	$\mu(e)$
Declaration(DataProperty (e))	$\texttt{CLA}(x_\alpha\ mo{:}DataPropertyDeclaration)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$	$\mu(e)$
Declaration(Individual (e))	$\texttt{CLA}(x_\alpha\ mo{:}IndividualDeclaration)$ $\texttt{OPA}(mo{:}entity\ x_\alpha\ x_e)$	$\mu(e)$

Table A.6: Translation of Data Ranges

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
DataComplementOf(dr)	$\texttt{CLA}(x_\alpha\ mo{:}DataComplementOf)$ $\texttt{OPA}(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(dr)$
DataOneOf$(\mathbf{ct}_1 \ldots \mathbf{ct}_n)$	$\texttt{CLA}(x_\alpha\ mo{:}DataOneOf)$ $\texttt{DPA}(mo{:}constants\ x_\alpha\ \mathbf{ct}_1)$ $\ldots$ $\texttt{DPA}(mo{:}constants\ x_\alpha\ \mathbf{ct}_n)$	$\emptyset$
DatatypeRestriction$(dr\ facet\ \mathbf{ct})$	$\texttt{CLA}(x_\alpha\ mo{:}DatatypeRestriction)$ $\texttt{OPA}(mo{:}dataRange\ x_\alpha\ x_{dr})$ $\texttt{DPA}(mo{:}restrictionValue\ x_\alpha\ \mathbf{ct})$ $\texttt{DPA}(mo{:}facetType\ x_\alpha\ facet)$	$\mu(dr)$

Table A.7: Translation of Object Property Expressions

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
InverseObjectProperty(op)	CLA$(x_\alpha$ *mo:InverseObjectProperty*	$\mu(op)$

Table A.8: Translation of Boolean Concepts

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
ObjectUnionOf$(c_1 \ldots c_n)$	CLA$(x_\alpha$ *mo:ObjectUnionOf*$)$ OPA$($*mo:classes* $x_\alpha\ x_{c_1})$ ... OPA$($*mo:classes* $x_\alpha\ x_{c_n})$	$\bigcup_{i=1}^{n} \mu(c_i)$
ObjectIntersectionOf$(c_1 \ldots c_n)$	CLA$(x_\alpha$ *mo:ObjectIntersectionOf*$)$ OPA$($*mo:classes* $x_\alpha\ x_{c_1})$ ... OPA$($*mo:classes* $x_\alpha\ x_{c_n})$	$\bigcup_{i=1}^{n} \mu(c_i)$
ObjectComplementOf(c)	CLA$(x_\alpha$ *mo:ObjectComplementOf*$)$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(c)$
ObjectOneOf$(in_1 \ldots in_n)$	CLA$(x_\alpha$ *mo:ObjectOneOf*$)$ OPA$($*mo:individuals* $x_\alpha\ in_1)$ ... OPA$($*mo:individuals* $x_\alpha\ in_n)$	$\bigcup_{i=1}^{n} \mu(in_i)$

Table A.9: Translation of Object Property Restrictions

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
ObjectSomeValuesFrom$(op\ c)$	CLA$(x_\alpha$ *mo:ObjectSomeValuesFrom*$)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(op) \cup \mu(c)$
ObjectAllValuesFrom$(op\ c)$	CLA$(x_\alpha$ *mo:ObjectAllValuesFrom*$)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(op) \cup \mu(c)$
ObjectExistsSelf(op)	CLA$(x_\alpha$ *mo:ObjectExistsSelf*$)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$	$\mu(op)$
ObjectHasValue$(op\ in)$	CLA$(x_\alpha$ *mo:ObjectHasValue*$)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:individual* $x_\alpha\ in)$	$\mu(op) \cup \mu(in)$
ObjectMaxCardinality$(n\ op\ c)$	CLA$(x_\alpha$ *mo:ObjectMaxCardinality*$)$ DPA$($*mo:cardinality* $x_\alpha\ n)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(op) \cup \mu(c)$
ObjectMinCardinality$(n\ op\ c)$	CLA$(x_\alpha$ *mo:ObjectMinCardinality*$)$ DPA$($*mo:cardinality* $x_\alpha\ n)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(op) \cup \mu(c)$
ObjectExactCardinality$(n\ op\ c)$	CLA$(x_\alpha$ *mo:ObjectExactCardinality*$)$ DPA$($*mo:cardinality* $x_\alpha\ n)$ OPA$($*mo:objectProperty* $x_\alpha\ x_{op})$ OPA$($*mo:class* $x_\alpha\ x_c)$	$\mu(op) \cup \mu(c)$

Table A.10: Translation of Datatype Property Restrictions

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
DataSomeValuesFrom$(dp\ dr)$	CLA$(x_\alpha\ mo{:}DataSomeValuesFrom)$ OPA$(mo{:}dataProperties\ x_\alpha\ x_{dp})$ DPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(dp) \cup \mu(dr)$
DataSomeValuesFrom$(dp_1\ \ldots\ dp_n\ dr)$	CLA$(x_\alpha\ mo{:}DataSomeValuesFrom)$ OPA$(mo{:}dataProperties$ $x_\alpha\ x_{L[dp_1\ \ldots\ dp_n]})$ DPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(L[dp_1\ \ldots\ dp_n]) \cup$ $\mu(dr)$
DataAllValuesFrom$(dp\ dr)$	CLA$(x_\alpha\ mo{:}DataAllValuesFrom)$ OPA$(mo{:}dataProperties\ x_\alpha\ x_{dp})$ DPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(dp) \cup \mu(dr)$
DataAllValuesFrom$(dp_1\ \ldots\ dp_n\ dr)$	CLA$(x_\alpha\ mo{:}DataAllValuesFrom)$ OPA$(mo{:}dataProperties$ $x_\alpha\ x_{L[dp_1\ \ldots\ dp_n]})$ DPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(L[dp_1\ \ldots\ dp_n]) \cup$ $\mu(dr)$
DataHasValue$(dp\ \mathbf{ct})$	CLA$(x_\alpha\ mo{:}DataHasValue)$ OPA$(mo{:}dataProperty\ x_\alpha\ x_{dp})$ DPA$(mo{:}constant\ x_(\alpha)\ \mathbf{ct})$	$\mu(dp)$
DataMaxCardinality$(n\ dp\ dr)$	CLA$(x_\alpha\ mo{:}DataMaxCardinality)$ DPA$(mo{:}cardinality\ x_\alpha\ n)$ OPA$(mo{:}dataProperty\ x_\alpha\ x_{dp})$ OPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(dp) \cup \mu(dr)$
DataMinCardinality$(n\ dp\ dr)$	CLA$(x_\alpha\ mo{:}DataMinCardinality)$ DPA$(mo{:}cardinality\ x_\alpha\ n)$ OPA$(mo{:}dataProperty\ x_\alpha\ x_{dp})$ OPA$(mo{:}dataRange\ x\ x_{dr})$	$\mu(dp) \cup \mu(dr)$
DataExactCardinality$(n\ dp\ dr)$	CLA$(x_\alpha\ mo{:}DataExactCardinality)$ DPA$(mo{:}cardinality\ x_\alpha\ n)$ OPA$(mo{:}dataProperty\ x_\alpha\ x_{dp})$ OPA$(mo{:}dataRange\ x_\alpha\ x_{dr})$	$\mu(dp) \cup \mu(dr)$

Table A.11: Translation of Class Axioms

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
SubClassOf$(c_1\ c_2)$	CLA$(x_\alpha\ mo{:}SubClassOf)$ OPA$(mo{:}subClass\ x_\alpha\ x_{c_1})$ OPA$(mo{:}superClass\ x_\alpha\ x_{c_2})$	$\mu(c_1) \cup \mu(c_2)$
EquivalentClasses$(c_1 \ldots c_n)$	CLA$(x_\alpha\ mo{:}EquivalentClasses)$ OPA$(mo{:}classes\ x_\alpha\ x_{c_1})$ $\ldots$ OPA$(mo{:}classes\ x_\alpha\ x_{c_n})$	$\bigcup_{i=1}^{n} \mu(c_i)$
DisjointClasses$(c_1 \ldots c_n)$	CLA$(x_\alpha\ mo{:}DisjointClasses)$ OPA$(mo{:}classes\ x_\alpha\ x_{c_1})$ $\ldots$ OPA$(mo{:}classes\ x_\alpha\ x_{c_n})$	$\bigcup_{i=1}^{n} \mu(c_i)$
DisjointUnion$(c\ c_1 \ldots c_n)$	CLA$(x_\alpha\ mo{:}DisjointUnion)$ OPA$(mo{:}class\ x_\alpha\ x_c)$ OPA$(mo{:}disjointClasses\ x_\alpha\ x_{c_1})$ $\ldots$ OPA$(mo{:}disjointClasses\ x_\alpha\ x_{c_n})$	$\mu(c) \cup \bigcup_{i=1}^{n} \mu(c_i)$

Table A.12: Translation of Object Property Axioms

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
SubObjectPropertyOf(op_1 op_2)	CLA(x_α *mo:SubObjectPropertyOf*) OPA(*mo:subObjectProperties* x_α x_{op_1}) OPA(*mo:superObjectProperty* x_α x_{op_2})	$\mu(op_1) \cup \mu(op_2)$
SubObjectPropertyOf(SubObjectPropertyChain(op_1 ... op_n) op)	CLA(x_α *mo:SubObjectPropertyOf*) OPA(*mo:subObjectProperties* x_α $x_{L[op_1 \ldots op_n]}$) OPA(*mo:superObjectProperty* x_α x_{op})	$\mu(op) \cup$ $\mu(L[op_1 \ldots op_n])$
EquivalentObjectProperties($op_1 \ldots op_n$)	CLA(x_α *mo:EquivalentObjectProperties*) OPA(*mo:objectProperties* x_α x_{op_1}) ... OPA(*mo:objectProperties* x_α x_{op_n})	$\bigcup_{i=1}^{n} \mu(op_i)$
DisjointObjectProperties($op_1 \ldots op_n$)	CLA(x_α *mo:DisjointObjectProperties*) OPA(*mo:objectProperties* x_α x_{op_1}) ... OPA(*mo:objectProperties* x_α x_{op_n})	$\bigcup_{i=1}^{n} \mu(op_i)$
ObjectPropertyDomain(op c)	CLA(x_α *mo:ObjectPropertyDomain*) OPA(*mo:objectProperty* x_α x_{op}) OPA(*mo:domain* x_α x_{op})	$\mu(op) \cup \mu(c)$
ObjectPropertyRange(op c)	CLA(x_α *mo:ObjectPropertyRange*) OPA(*mo:objectProperty* x_α x_{op}) OPA(*mo:range* x_α x_{op})	$\mu(op) \cup \mu(c)$
InverseObjectProperties(op_1 op_2)	CLA(x_α *mo:InverseObjectProperties*) OPA(*mo:objectProperty1* x_α x_{op_1}) OPA(*mo:objectProperty2* x_α x_{op_2})	$\mu(op_1) \cup \mu(op_2)$
TransitiveObjectProperty(op)	CLA(x_α *mo:TransitiveObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
FunctionalObjectProperty(op)	CLA(x_α *mo:FunctionalObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
InverseFunctionalObjectProperty(op)	CLA(x_α *mo:InverseFunctionalObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
ReflexiveObjectProperty(op)	CLA(x_α *mo:ReflexiveObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
IrreflexiveObjectProperty(op)	CLA(x_α *mo:IrreflexiveObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
SymmetricObjectProperty(op)	CLA(x_α *mo:SymmetricObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$
AntisymmetricObjectProperty(op)	CLA(x_α *mo:AntisymmetricObjectProperty*) OPA(*mo:objectProperty* x_α x_{op})	$\mu(op)$

Table A.13: Translation of Data Property Axioms

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
SubDataPropertyOf(dp_1 dp_2)	CLA(x_α *mo:SubDataPropertyOf*) OPA(*mo:subDataProperty* x_α x_{dp_1}) OPA(*mo:superDataProperty* x_α x_{dp_2})	$\mu(dp_1) \cup \mu(dp_2)$
EquivalentDataProperties($dp_1 \ldots dp_n$)	CLA(x_α *mo:EquivalentDataProperties*) OPA(*mo:dataProperties* x_α x_{dp_1}) ... OPA(*mo:dataProperties* x_α x_{dp_n})	$\bigcup_{i=1}^{n} \mu(dp_i)$
DisjointDataProperties($dp_1 \ldots dp_n$)	CLA(x_α) *mo:DisjointDataProperties*) OPA(*mo:dataProperties* x_α x_{dp_1}) ... OPA(*mo:dataProperties* x_α x_{dp_n})	$\bigcup_{i=1}^{n} \mu(dp_i)$
DataPropertyDomain(dp c)	CLA(x_α *mo:DataPropertyDomain*) OPA(*mo:dataProperty* x_α x_{dp}) OPA(*mo:domain* x_α x_c)	$\mu(dp) \cup \mu(c)$
DataPropertyRange(dp dr)	CLA(x_α *mo:DataPropertyRange*) OPA(*mo:objectProperty* x_α x_{dp}) OPA(*mo:range* x_α x_{dr})	$\mu(dp) \cup \mu(dr)$
FunctionalDataProperty(dp)	CLA(x_α *mo:FunctionalDataProperty*) OPA(*mo:dataProperty* x_α x_{op})	$\mu(dp)$

Table A.14: Translation of Assertions

α	$\Lambda(\alpha)$	$\Xi(\alpha)$
ClassAssertion$(in\ c)$	CLA$(x_\alpha\ mo{:}ClassAssertion)$ OPA$(mo{:}individual\ x_\alpha\ in)$ OPA$(mo{:}description\ x_\alpha\ x_c)$	$\mu(in) \cup$ $\mu(c)$
ObjectPropertyAssertion$(op\ in_1\ in_2)$	CLA$(x_\alpha\ mo{:}ObjectPropertyAssertion)$ OPA$(mo{:}objectProperty\ x_\alpha\ x_{op})$ OPA$(mo{:}sourceIndividual\ x_\alpha\ in_1)$ OPA$(mo{:}targetIndividual\ x_\alpha\ in_2)$	$\mu(op) \cup$ $\mu(in_1) \cup$ $\mu(in_2)$
NegativeObjectPropertyAssertion$(op\ in_1\ in_2)$	CLA$(x_\alpha$ $mo{:}NegativeObjectPropertyAssertion)$ OPA$(mo{:}objectProperty\ x_\alpha\ x_{op})$ OPA$(mo{:}sourceIndividual\ x_\alpha\ in_1)$ OPA$(mo{:}targetIndividual)\ x_\alpha\ in_2)$	$\mu(op) \cup$ $\mu(in_1) \cup$ $\mu(in_2)$
SameIndividual$(in_1 \ldots in_n)$	CLA$(x_\alpha)\ mo{:}SameIndividual)$ OPA$(mo{:}individuals)\ x_\alpha\ in_1)$ ... OPA$(mo{:}individuals)\ x_\alpha\ in_n)$	$\bigcup_{i=1}^{n} \mu(in_i)$
DifferentIndividuals$(in_1 \ldots in_n)$	CLA$(x_\alpha)\ mo{:}DifferentIndividuals)$ OPA$(mo{:}individuals)\ x_\alpha\ in_1)$ ... OPA$(mo{:}individuals)\ x_\alpha\ in_n)$	$\bigcup_{i=1}^{n} \mu(in_i)$
DataPropertyAssertion$(dp\ in\ \mathbf{ct})$	CLA$(x_\alpha\ mo{:}DataPropertyAssertion)$ OPA$(mo{:}dataProperty)\ x_\alpha\ x_{op})$ OPA$(mo{:}sourceIndividual)\ x_\alpha\ in)$ DPA$(mo{:}targetValue)\ x_\alpha\ \mathbf{ct})$	$\mu(op) \cup$ $\mu(in)$
NegativeDataPropertyAssertion$(dp\ in\ \mathbf{ct})$	CLA$(x_\alpha$ $mo{:}NegativeDataPropertyAssertion)$ OPA$(mo{:}dataProperty)\ x_\alpha\ x_{op})$ OPA$(mo{:}sourceIndividual)\ x_\alpha\ in)$ DPA$(mo{:}targetValue)\ x_\alpha\ \mathbf{ct})$	$\mu(dp) \cup$ $\mu(in)$

Bibliography

[AM97] P. Smets A. Motro. *Uncertainty Management In Information Systems*. Springer, 1997.

[BCRS06] P. Buitelaar, P. Cimiano, S. Racioppa, and M. Siegel. Ontology-based information extraction with soba. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

[BDP92] Salem Benferhat, Didier Dubois, and Henri Prade. Representing default rules in possibilistic logic. In *KR*, pages 673–684, 1992.

[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.

[BLP04] S. Benferhat, S. Lagrue, and O. Papini. Reasoning with partially ordered information in a possibilistic logic framework. *Fuzzy Sets and Systems*, 144(1):25–41, 2004.

[BOS03] P. Buitelaar, D. Olejnik, and M. Sintek. OntoLT: A protégé plug-in for ontology extraction from text. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2003.

[CGL+06] Carola Catenacci, Aldo Gangemi, Jos Lehmann, Malvina Nissim, and Valentina Presutti. Design rationales for collaborative development of networked ontologies - state of the art and the collaborative ontology design ontology. Technical report, CNR; NeOn Deliverable D2.1.1, 2006.

[Cir01] F. Ciravegna. Adaptive information extraction from text by rule induction and generalization. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1251–1256, 2001.

[CV05] Philipp Cimiano and Johanna VèŮŻlker. Text2onto - a framework for ontology learning and data-driven change discovery. In Andres Montoyo, Rafael Munoz, and Elisabeth Metais, editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238, Alicante, Spain, JUN 2005. Springer.

[DLP94] Didier Dubois, Jérôme Lang, and Henri Prade. Possibilistic logic. In *Handbook of logic in Aritificial Intelligence and Logic Programming*, pages 439–513, 1994.

[DP91] Didier Dubois and Henri Prade. Epistemic entrenchment and possibilistic logic. *Artif. Intell.*, 50(2):223–239, 1991.

[DP98] Didier Dubois and Henri Prade. Possibility theory: qualitative and quantitative aspects. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 169–226, 1998.

[DP04] Z. Ding and Y. Peng. A Probabilistic Extension to Ontology Language OWL. In *Proceedings of the 37th Hawaii International Conference On System Sciences (HICSS-37)*., Big Island, Hawaii, January 2004.

[DS05] Michael Dürig and Thomas Studer. Probabilistic abox reasoning: Preliminary results. In *Description Logics*, 2005.

[Fel98] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[FGM07] Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. Ontogen: Semi-automatic ontology editor. In *HCI (9)*, pages 309–318, 2007.

[FK00] F. Freitag and N. Kushmerick. Boosted Wrapper Induction. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 577–583, 2000.

[FN98] D. Faure and C. Nedellec. A corpus-based conceptual clustering method for verb frames and ontology. In *Proceedings of the LREC Workshop on Adapting lexical and corpus resources to sublanguages and applications*, 1998.

[GL02] Rosalba Giugno and Thomas Lukasiewicz. P-shoq(d): A probabilistic extension of shoq(d) for probabilistic ontologies in the semantic web. In *JELIA*, pages 86–97, 2002.

[HBP+07] Peter Haase, Saartje Brockmans, Raul Palma, Jérôme Euzenat, and Mathieu d'Aquin. Updated version of the networked ontology model. Technical Report D1.1.2, University of Karlsruhe, AUG 2007.

[Hea92] M.A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 539–545, 1992.

[Hei94] Jochen Heinsohn. Probabilistic description logics. In *UAI'94*, pages 311–318, 1994.

[HHR+06] Peter Haase, Pascal Hitzler, Sebastian Rudolph, Guilin Qi, Marko Grobelnik, Igor Mozetič, Damjan Bojadžiev, Jérôme Euzenat, Mathieu d'Aquin, Aldo Gangemi, and Carola Catenacci. D3.1.1 context languages - state of the art. Technical Report D3.1.1, Universität Karlsruhe, August 2006.

[HM05] P. Haase and B. Motik. A Mapping System for the Integration of OWL-DL Ontologies. In *In Proceedings of the ACM-Workshop: Interoperability of Heterogeneous Information Systems (IHIS05)*, November 2005.

[HV05] Peter Haase and Johanna Völker. Ontology learning and reasoning - dealing with uncertainty and inconsistency. In Paulo C. G. da Costa, Kathryn B. Laskey, Kenneth J. Laskey, and Michael Pool, editors, *Proceedings of the Workshop on Uncertainty Reasoning for the Semantic Web (URSW)*, pages 45–55, NOV 2005.

[Jae94] Manfred Jaeger. Probabilistic reasoning in terminological logics. In *KR'94*, pages 305–316, 1994.

[KLP97] Daphne Koller, Alon Y. Levy, and Avi Pfeffer. P-classic: A tractable probablistic description logic. In *Proc. of AAAI'97*, pages 390–397, 1997.

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in owl ontologies. In *ESWC'06*, pages 170–184, 2006.

[Mot07] Boris Motik. On the Properties of Metamodeling in OWL. *Journal of Logic and Computation*, 17(4):617–637, 2007.

[MST07] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Repairing ontology mappings. In *Proc. of AAAI'07*, pages 1408–1413, 2007.

[NF04] Henrik Nottelmann and Norbert Fuhr. pdaml+oil: A probabilistic extension to daml+oil based on probabilistic datalog. In *Proceedings Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 227–234, 2004.

[NVCN04] R. Navigli, P. Velardi, A. Cucchiarelli, and F. Neri. Extending and enriching WordNet with OntoLearn. In *Proc. of the GWC 2004*, pages 279–284, 2004.

[POT70] Chaim Perelman and Lucie Olbrechts-Tyteca. *TraitåžŘéIJš de l'argumentation: La nouvelle rhåžŘéIJštorique*. Paris: Presses Universitaires de France, 1970.

[PST04] Helena Sofia Pinto, Steffen Staab, and Christoph Tempich. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In *Proc. of ECAI'04*, pages 393–397, 2004.

[QHJ07] Guilin Qi, Peter Haase, and Qiu Ji. D1.2.1 consistency models for networked ontologies. Technical Report D1.2.1, Universität Karlsruhe, FEB 2007.

[SSW05] L. Serafini, H. Stuckenschmidt, and H. Wache. A Formal Investigation of Mapping Languages for Terminological Knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI05*, Edinburgh, UK, August 2005.

[Tem06] Christoph Tempich, editor. *Ontology Engineering and Routing in Distributed Knowledge Management Applications*. 2006.

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

[Tou58] S. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.

[TPSS05] Christoph Tempich, Helena Sofia Pinto, York Sure, and Steffen Staab. An argumentation ontology for distributed, loosely-controlled and evolving engineering processes of ontologies (diligent). In *Proc. of ESWC'06*, pages 241–256, 2005.