NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D3.2.3 Methods for contextual reasoning

Deliverable Co-ordinator: **Guilin Qi**

Deliverable Co-ordinating Institution: **UKARL**

Other Authors: **Klaas Dellschaft (UKOBL); Qiu Ji (UKARL)**

In our previous deliverable D3.1.3, we provided a common formalism for representing context of ontologies. We instantiated our generic definition of context for three specific forms of context: *Provenance*, *Arguments* and *mapping*. In this deliverable, we will further our work in previous deliverable by considering methods for contextual reasoning. We first present a contextual framework for diagnosis and repair. In our framework, we use the RaDON system to debug and diagnose inconsistencies and use the Cicero system to provide a discussion forum to the users. By integrating Cicero with RaDON, the users can utilize the information obtained by discussion in Cicero as context information to come up with a good repair strategy. We then present possibilistic description logics which extend description logics to deal with uncertainty. We show how possibilistic description logics provide an integrated framework to reason with mappings and uncertainty.

Document Identifier:	NEON/2009/D3.2.3/v1.0	Date due:	July 30, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 14, 2009
Project start date	March 1, 2006	Version:	v1.0
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 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 adress: 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	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**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
- UKobl
- Partner X

Change Log

Version	Date	Amended by	Changes
0.1	31-03-2009	Guilin Qi	Create the deliverable
0.2	19-05-2009	Klaas Dellschaft	Introduced the chapter about Cicero and extended the chapter about integrating Cicero and RaDON.
0.3	31-07-2009	Guilin Qi, Qiu Ji	Complete the deliverable
0.4	14-08-2009	Guilin Qi, Qiu Ji	Address comments from NeOn review

Executive Summary

In our previous deliverable D3.1.3, we provided a common formalism for representing context of ontologies. We instantiated our generic definition of context for three specific forms of context: *Provenance*, *Arguments* and *mapping*.

In this deliverable, we will further our work in previous deliverable by considering methods for contextual reasoning. we present our framework which helps to diagnose and repair collaboratively constructed ontologies. Our framework integrates RaDON, an ontology diagnosis and repair tool, and Cicero, which provides discussion functionality for the ontology developers. In order to capture the changes on the ontologies proposed by RaDON, the user needs to set up a Oyster server. Oyster is a central registry for ontologies which allows for exchanging ontologies, ontology changes and further ontology related metadata. In our case, Oyster is used to capture ontology changes and to propagate them to the other collaborating devlopers. In our framework, we use the RaDON system to debug and diagnose inconsistencies and use the Cicero system to provide a discussion forum to the users. By integrating Cicero with RaDON, the users can utilize the information obtained by discussion in Cicero as context information to come up with a good repair strategy.

We then present possibilistic description logics which extend description logics to deal with uncertainty and show how possibilistic description logics provide an integrated framework to reason with mappings and uncertainty. We give semantics and syntax of possibilistic description logics. We then define some inference services in possibilistic description logics. We show how inference services in possibilistic description logics can be reduced to the problem of computing the *inconsistency degree* of the possibilistic description logic-based knowledge base. Since possibilistic inference suffers from the *drowning problem*, which means that all the axioms whose certainty degrees are lower than the inconsistency degree of the possibilistic description logic-based knowledge base cannot contribute to the inference, we consider a drowning-free variant of possibilistic inference, called linear order inference. We provide an algorithm to compute the inconsistency degree of a possibilistic description logic-based knowledge base and an algorithm for linear ordered inference. Finally, we implement the algorithms and provide a plugin in NeOn toolkit.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Role of WP3 in 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 Methods for Contextual Reasoning

In our previous deliverable D3.1.3, we provided a common formalism for representing context of ontologies. We instantiated our generic definition of context for three specific forms of context: *Provenance*, *Arguments* and *mapping*.

In this deliverable, we will further our work in previous deliverable by considering methods for contextual reasoning. We first present a contextual framework for diagnosis and repair. Before presenting the framework, we provide motivation for integrating RaDON and Cicero to show that they can complement each other. We then integrate RaDON and Cicero through NeOn Toolkit and propose an enhanced ontology editing framework. In order to capture the changes on the ontologies proposed by RaDON, the user needs to set up a Oyster server. Oyster is a central registry for ontologies which allows for exchanging ontologies, ontology changes and further ontology related metadata. In our case, Oyster is used to capture ontology changes and to propagate them to the other collaborating devlopers. In our framework, we use the RaDON system to debug and diagnose inconsistencies and use the Cicero system to provide a discussion forum to the users. By integrating Cicero with RaDON, the users can utilize the information obtained by discussion in Cicero as context information to come up with a good repair strategy.

We then present possibilistic description logics which extend description logics to deal with uncertainty and show how possibilistic description logics provide an integrated framework to reason with mappings and un-

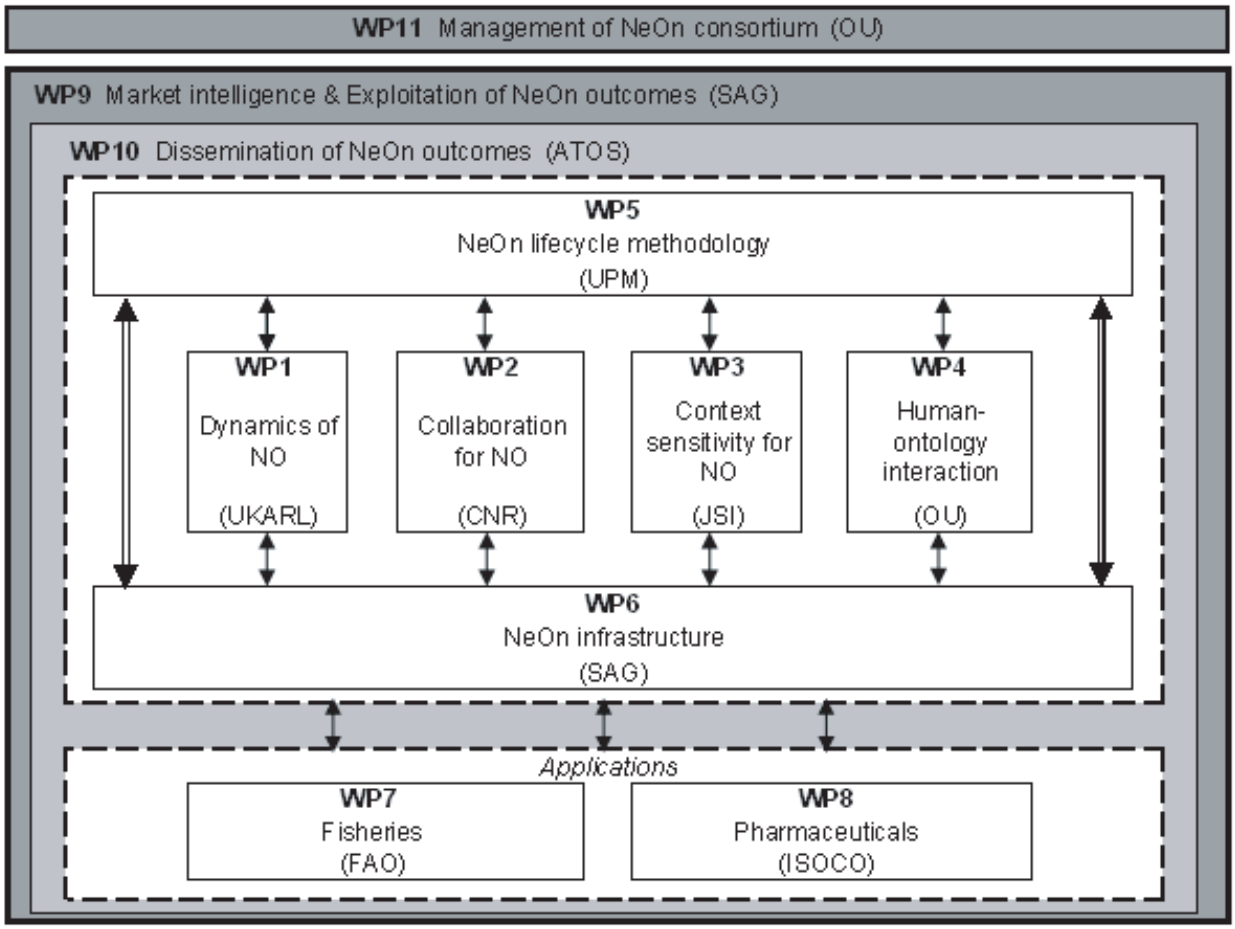

Figure 1.1: Relationships between different workpackages in NeOn

certainty. We give semantics and syntax of possibilistic description logics. We then define some inference services in possibilistic description logics. We show how inference services in possibilistic description logics can be reduced to the problem of computing the *inconsistency degree* of the description logic-based knowledge base. Since possibilistic inference suffers from the *drowning problem*, we consider a drowning-free variant of possibilistic inference, called linear order inference. We provide an algorithm to compute the inconsistency degree of a description logic-based and an algorithm for linear ordered inference. Finally, we implement the algorithms and developed a plugin for NeOn toolkit.

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 uncertainty information which can be handled by our possibilistic description logics. In deliverable D3.3.1, the Alignment Server is given as the infrastructure for contextualizing ontologies by finding relations that it has with other ontologies. In our deliverable, we provide possibilistic description logics for reasoning with distributed ontologies connected by mappings.

1.3 Overview of the Deliverable

In this deliverable we first introduce RaDON system and Cicero system in Chapter 2 and Chapter 3 respectively. We then provide a contextual framework for diagnosis and repair in Chapter 4. In Chapter 5, we present possibilistic description logics to reasoning with mappings and uncertainty information. We conclude this deliverable in Chapter 6.

Chapter 2

The RaDON System

RaDON[1] has been developed to deal with inconsistency[2] and incoherence[3] for ontology networks. It supports OWL-DL and is implemented in Java as a plug-in for the NeOn toolkit[4], which is an extensible ontology engineering environment to handle multiple networked ontologies. The RaDON plugin has already been applied in the FAO[5] case study in the context of diagnosing and repairing automatically learned ontologies. Results of these applications have been reported in NeOn Deliverable D1.2.2 [QHJV07].

In this section, we first introduce the functionalities of the RaDON system and then we present the process to debug and repair ontology networks. Afterwards, we describe the two new algorithms respectively: relevance-directed algorithm and paraconsistency-based algorithm for debugging ontologies.

2.1 Functionalities of RaDON

RaDON provides a set of techniques for dealing with inconsistency and incoherence in ontologies. In particular, RaDON supports novel strategies and consistency models for distributed and networked environments.

RaDON extends the capabilities of existing reasoners with the functionalities to deal with inconsistency and incoherence. Specifically, the functionalities provided by RaDON include: (1) debugging an incoherent or inconsistent ontology to explain why a concept is unsatisfiable or why the ontology is inconsistent, (2) repairing an ontology automatically by computing all possible explanations w.r.t. all unsatisfiable concepts if the ontology is incoherence, or w.r.t. the inconsistent ontology if it is inconsistent, (3) repairing an ontology manually based on the debugging results. For the manual repair, the user can choose the axioms to be removed for restoring the coherence or consistency. (4) coping with inconsistency based on a paraconsistency-based algorithm.

2.2 Process to Debug and Repair Ontology Networks

The plug-in can be used to diagnose and repair not only a single ontology, but also multiple ontologies that are networked. In particular, we consider ontologies that are networked via *mappings*. Mappings essentially are correspondences between the elements of two different ontologies, in the most simple case in the form of subclassOf or equivalentClasses axioms (cf. [HBP+07b] for a definition of the networked ontology model). The mapping assertions may additionally be annotated with confidence values.

RaDON takes as input an ontology network consisting of ontologies and mappings between the ontologies. For our definition of inconsistency in the ontology network, we follow a global model semantics: An ontol-

[1] http://radon.ontoware.org/demo.htm

[2] An ontology is inconsistent iff it has no model.

[3] An ontology is incoherent iff it contains at least one unsatisfiable concept in its terminology. A concept is unsatisfiable if it is mapped to the empty set in all models of the ontology.

[4] http://www.neon-toolkit.org/

[5] http://www.fao.org/

ogy network is inconsistent (resp. incoherent) if the ontology obtained by merging the ontologies and their mappings is inconsistent (resp. incoherent). For the repair process we focus on repairing the mappings, assuming the ontologies are already individually coherent and consistent. Consequently only axioms in the mappings could be removed to resolve incoherence or inconsistency.

If the merged ontology is incoherent, the system computes all the unsatisfiable concepts. When an unsatisfiable concept is selected, all the minimal unsatisfiability-preserving subsets (MUPS)[6] w.r.t. this concept are computed. Similarly, inconsistencies in the ontology network are debugged: For inconsistencies, all the minimal inconsistent subsets (MIS)[7] w.r.t. the ontology are computed.

After obtaining the debugging results, we can repair the ontology automatically according to the proposed solution by our system. Also, the user can repair the ontology by manually choosing the axioms to be removed. To help the user to make a decision, our system provides the confidence values or scores for each axiom which could be removed to resolve incoherence or inconsistency. See [QHJV07] for more details.

2.3 Relevance-directed Algorithm

A key problem of diagnosing and repairing ontology networks is to compute all or some MUPS for an unsatisfiable concept efficiently. The RaDON system provides various strategies to compute some or all MUPS based on the relevance-directed algorithm [JQH08]. This algorithm incrementally selects sub-ontologies using a *selection function* and finds a set of MUPS from these sub-ontologies for an unsatisfiable concept. Our algorithm is adapted from the algorithm given in [KPHS07] which is based on Reiter's Hitting Set Tree (HST) algorithm [Rei87].
Specifically, the relevance-directed algorithm provides the following strategies when computing MUPS for an unsatisfiable concept:

- Compute one MUPS;
- Compute all MUPS and all hitting sets[8] for the set of all MUPS;
- Compute some (not all) MUPS and some hitting sets for the set of all MUPS.

Therefore, the users could choose different strategies according to the ontology and their purpose. For example, if the testing ontology is relatively small, they could try to compute all MUPS. Also, if the users only intend to resolve the incoherence with some but not all solutions, we can choose the third strategy.

[6] Let C be a named concept which is unsatisfiable in an ontology $\mathcal{O}$. A set $\mathcal{M}\subseteq\mathcal{O}$ is a *minimal unsatisfiability-preserving subset (MUPS)* [SC03] of $\mathcal{O}$ if C is unsatisfiable in $\mathcal{M}$, and C is satisfiable in every subset $\mathcal{M}' \subset \mathcal{M}$.

[7] Given an inconsistent ontology $\mathcal{O}$, a set $\mathcal{M}\subseteq\mathcal{O}$ is a *minimal inconsistent subset (MIS)* w.r.t. $\mathcal{O}$ if $\mathcal{M}$ is inconsistent and every subset $\mathcal{M}' \subset \mathcal{M}$ is consistent.

[8] Given a set $S = \{\mathcal{M}_1, ..., \mathcal{M}_n\}$ of MUPS of an ontology $\mathcal{O}$ for an unsatisfiable concept, a *hitting set* T for S is a subset of $\mathcal{O}$ such that $\mathcal{M}_i \cap T \neq \emptyset$ for all $1 \leq i \leq n$.

Chapter 3

The Cicero System

Creating and designing an ontology is a complex task that requires the collaboration of domain and ontology engineering experts. For coming to a consensual model of a domain that is expressed by an ontology, the participants in the engineering process must discuss their different viewpoints in an efficient manner. Furthermore, discussion may become important during resolving inconsistencies in ontologies, especially if there exist several possible solutions. Thus, discussions are an important part of collaborative ontology engineering.

In the context of NeOn, the Cicero tool has been developed that facilitates an asynchronous discussion and decision taking process between participants of an ontology engineering project. Two main objectives of capturing discussions in Cicero can be distinguished:

- *Higher efficiency*: Cicero supports its users in discussing the design rationale of ontologies. The whole discussion including the pro and contra arguments is recorded, leading to fewer redundancies in disputes. It has been shown that the applied discussion methodology facilitates efficient discussions and accelerates convergence to a solution.
- *Enhanced documentation*: The captured discussions reflect the design rationale of an ontology. By attaching a discussion to the entities in the ontology, it is possible later to understand why certain elements are modeled as they are. Furthermore, prior discussions can easily be resumed if e. g. new requirements have to be taken into account.

The first objective is accomplished by the Cicero tool itself.[1] Details about its underlying argumentation model and discussion workflow are described in section 3.1 and in [DGG+08]. The second objective is accomplished by integrating Cicero with the NeOn toolkit, which is described in section 3.2.

3.1 Cicero Argumentation Model

The argumentation model, that is underlying the Cicero tool, is based on the DILIGENT argumentation framework [PST04] and the Potts and Bruns model [PB88]. They are both extensions of the idea of the Issue Based Information Systems [KR70, RW73]. All these models help in structuring an *issue* or problem and to simultaneously derive possible solutions with the help of discussions. In the case of inconsistency reasoning, an issue might e. g. be how to resolve an inconsistency that was detected by RaDON.

The Cicero argumentation model combines the general structure for representing discussions from the DILIGENT argumentation framework with the idea from the Potts and Bruns model of annotating ontology elements and changes with the corresponding discussions. In Cicero, a discussion always starts with an issue that can be raised by either of the participants (i. e. ontology/knowledge engineers or users) e. g. after detecting an inconsistency with RaDON. Subsequently, solutions are proposed and discussed. After some time, it

[1] It can be downloaded from `http://isweb.uni-koblenz.de/Research/Cicero/`. A running instance of Cicero is available at `http://cicero.uni-koblenz.de`.

Figure 3.1: Prototypical workflow for discussions about the design rationale.

has to be decided which of the proposed solutions will be implemented in the ontology (see Fig. 3.1). More details about Cicero's argumentation model and the Cicero wiki are available in [DGG+08].

3.2 Integration with NeOn Toolkit

In order to better support the ontology engineering process, there currently exist three different plugins that integrate functionality of the Cicero wiki into the NeOn toolkit:

1. **Cicero OWL**: Annotating, creating and retrieving discussions related to elements in OWL ontologies.
2. **Cicero OWL Changes**: Annotating creating and retrieving discussions related to adding, removing and editing of elements in OWL ontologies. Integrates Cicero functionality with the *Change Capturing plugin*[2] .
3. **Cicero FLogic**: Annotating, creating and retrieving discussions related to elements in FLogic ontologies.

All three plugins allow for an easy creation of new discussions and for searching existing discussions from within the NeOn toolkit. For example, if a user of the NeOn toolkit wants to create a new discussion he may select the ontology entities in the environment that should be discussed. Then the corresponding plugin creates a new discussion in the Cicero wiki and annotates the selected ontology entities with the URL of the discussion. The annotation can then be used at a later time to retrieve discussions that are related to specific parts of the ontology.

All above mentioned plugins are available for download on the corresponding page in NeOn toolkit wiki[3] and on the Update Site of the NeOn toolkit that can be directly accessed from within the toolkit. The plugins directly ship with the latest documentation of their functionality which is then available in the help system of the toolkit. Alternatively, the documentation can also be accessed on the page in the NeOn toolkit wiki.

3.3 Conclusions

Altogether, Cicero helps with its IBIS-based argumentation framework to have better structured and more efficient discussions. Furthermore, the integration with the NeOn toolkit leads to a better support of the ontology engineering lifecycle in which discussions about the design rationale of ontology elements play an inherent role. Finally, Cicero and its integration with the NeOn toolkit reduces the required effort for establishing the provenance links between design rationale discussions and the affected ontology elements.

[2] http://www.neon-toolkit.org/wiki/index.php/Change_Capturing
[3] http://www.neon-toolkit.org/wiki/index.php/Cicero

Chapter 4

A Contextual Framework for Diagnosis and Repair

In this chapter, we propose a framework for networked ontology diagnosis and repair by tightly integrating Cicero and RaDON. We first give motivation of our work, then propose our framework. This framework has been implemented as a plugin for NeOn toolkit.

4.1 Motivation

In this section, we discuss the motivation to propose a contextual framework for diagnosis and repair.

According to NeOn deliverable D3.1.3, argumentation structures are a form of context that is obtained in collaborative ontology engineering processes. 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. It is the core of the Cicero framework given in NeOn deliverable D2.3.1 and chapter 3. Some initial experiments have been conducted. Another framework currently under development under the Neon Project is C-ODO [CGL+06]. We have instantiated the generic definition of context by argumentation structure in Section 4.2.2 in D3.1.3.

As described in chapter 3, Cicero is a wiki based solution that supports discussions during a collaborative ontology development project. A typical problem is that conflicting statements may be introduced during collaboratively editing an ontology. The main reason is that an ontology is too complex for foreseeing all possible side-effects of an ontology change.

Several of these side-effects may be detected by discussing the issues and possible solutions with other ontology developers. This especially helps in coming to a consensual understanding of how to best implement a change and how it relates to the already existing ontology. But nevertheless, in very complex or networked ontologies it is impossible for a human to foresee all logical side-effects of a proposed solution that should be implemented and thus during implementing the changes, conflicting or inconsistent statements may be introduced into the ontology.

But even more problematic is the case, when there was no discussion between the developers prior to applying a change. A typical problem in online collaboration is that without good motivation, people are typically reluctant to discuss with each other. A possible reason is that people perceive online discussions as a time-consuming process. Thus, for smaller issues they will not discuss their proposed solution and how to implement it in the ontology. This leads to a higher probability that logical side-effects are overseen and conflicting or inconsistent statements are introduced into the ontology.

In both cases, a debugging system like RaDON helps in detecting the conflicting or inconsistent statements. RaDON provides justifications for the detected conflicts and inconsistencies. Furthermore, it may provide possible solutions for resolving the inconsistencies. By integrating RaDON with Cicero, we may directly feed back the spotted inconsistencies, the justifications and solution proposals of RaDON into the discussions

held in Cicero. There, the ontology developers can then discuss how to cope with the inconsistency and whether they can adapt one of the solutions proposed by RaDON or whether none of the proposed solutions is appropriate. Therefore, we can enhance the functionality of Cicero with the help of RaDON.

But not only Cicero and the discussions between the developers may benefit from integration with RaDON. Also a debugging system like RaDON may use information stored in already existing discussions. This information can be used as context information during computing the diagnosis from all minimal unsatisfiability-preserving subsets (MUPS) for an unsatisfiable concept in an ontology (see Chapter 2).

A single diagnosis of RaDON consists of axioms selected from the different MUPS. For example, if we have n MUPS for an unsatisfiable concept, each of them contains m_i $(i = 1, ..., n)$ axioms, then there are $m_1 \times .... \times m_n$ different possible diagnoses. In the worst case, RaDON needs to check all of these diagnoses to find a preferred one. Thus, computing all possible diagnoses in RaDON is not very efficient when the number of MUPS is large and the size of each MUPS is big.

A second example is when it comes to repairing an inconsistent ontology. In that case, RaDON typically suggests to remove some axioms from the ontology. However, this removal may not be desirable because the inconsistency may be caused by only parts of an axiom or by a wrong definition of a concept.

In both cases, Cicero may help in semi-automatically solving the problem: For example, in a first step we may ask the user of RaDON to exclude those axioms from the different MUPS that seem to be correct and thus shouldn't be removed from the ontology. For this purpose, RaDON may show the user the discussions that are attached to the different axioms. This exclusion of axioms from the MUPS can drastically reduce the search space of RaDON for finding a good diagnosis. For example, if there are 3 MUPS which each contain 4 axioms then there are 64 possible diagnoses. If the user is able to exclude 2 axioms from each of the MUPS, then the search space is already reduced to 8 possible diagnoses.

In a second step, the user may decide whether it is possible to weaken some of the axioms in the MUPS instead of completely removing them from the ontology. Again, this decision of the user may be based on the discussions stored in Cicero. Thus, in both steps RaDON may support the decisions of the user by showing him or her all discussions related to the axioms contained in the MUPS.

In both steps, we exploit context information provided by Cicero to deal with inconsistency, thus influence the result of diagnosis and repair.

4.2 Our Framework

We integrate RaDON and Cicero through NeOn Toolkit and propose an enhanced ontology editing framework (see Figure 4.1). In order to capture the changes on the ontologies proposed by RaDON, the user needs to set up a Oyster server. Oyster is a central registry for ontologies which allows for exchanging ontologies, ontology changes and further ontology related metadata. In our case, Oyster is used to capture ontology changes and to propagate them to the other collaborating developers. Our framework ensures that possible inconsistencies which were introduced during implementing ontology changes are quickly spotted and resolved in collaboration with the other developers of the ontology.

In our framework, a workflow always starts with a consistent ontology. This ontology may also be empty. For this ontology, first an issue is identified, e. g. the ontology should be extended with new elements or existing elements should be modified. If this issue is explicitly formulated, one may use its description to create an issue in Cicero and discuss possible solutions with the other collaborating developers (see Chapter 3). After the discussion, the developers decided on which of the possible solution proposals should be implemented in response to the issue.

But it is also possible that a developer skips the explicit discussion process in Cicero and thinks about possible solutions himself. Then the single developer thinks about possible solutions for the issue at hand and also decides alone which solution he wants to implement. The advantage of skipping the explicit discussion process in Cicero is that the developer may save time because he needs not to explicitly formulate the issue and the possible solution proposals. But of course this may also be seen as a disadvantage as it increases

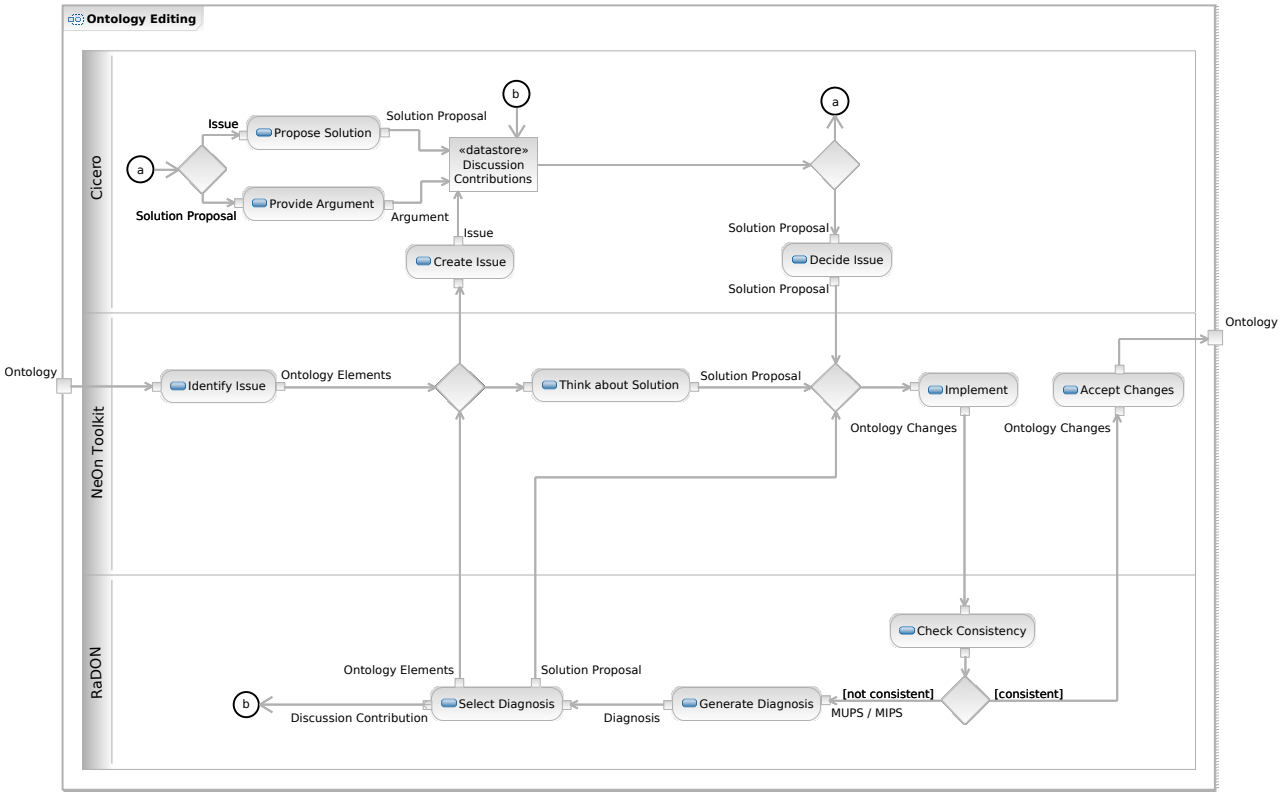

Figure 4.1: Workflow of activities in the enhanced ontology editing framework.

the probability that important side effects of the solution are overseen and that the whole process is not well documented.

In any case, after coming to a decision which solution should be implemented, the actual implementation of the changes in the ontology is done. These ontology changes are captured in Oyster. After finishing the implementation of the solution proposal, the user checks with the help of RaDON whether he introduced inconsistencies into the ontology.

If the ontology is consistent, the user accepts the ontology changes and they are propagated to the other developers of the ontology. Otherwise, RaDON will debug the inconsistencies by computing MUPS or MIPS. The MUPS and MIPS are then used by RaDON to generate a set of diagnosis and to help the user in spotting the ontology elements which cause the inconsistency. It is even possible, that RaDON automatically generates solution proposals which would resolve the inconsistencies. Furthermore, RaDON shows the user the discussions in Cicero which are related to the ontology elements contained in the diagnosis. The discussions may be used by the user and/or RaDON to decide which of the ontology elements in the MUPS/MIPS caused the inconsistency and how to resolve it.

Given the output of RaDON, the ontology developer has several choices. For example, he may select one of the automatically generated solution proposals and implement them in the ontology. Furthermore, if there already exists a discussion in Cicero for the current issue, the user may attach the diagnosis of RaDON and/or the possible solution proposals as new contributions to the existing discussion. Finally, if the ontology developer previously skipped the discussion process, he may also take the ontology elements contained in the MUPS and MIPS to either create a discussion of the issue in Cicero now (the newly created discussion is then automatically attached to these ontology elements) or he may try to find a solution without the help of the other collaborating ontology developers.

4.3 Implementation

Our framework has been implemented in NeOn Toolkit by integrating RaDON plug-in, Cicero plug-in and Oyster plug-in. Specifically, Cicero is integrated into RaDON to annotate an axiom in a justification, which facilitates the users to discuss whether an axiom should be removed or not, or to provide the reasons for future discussion. Integrating Cicero to Oyster is to provide the functionality to capture the changes made by RaDON and then annotate the changes.

In this section, we first talk about how to install our framework in NeOn Toolkit. Then we provide the details about how to use our framework.

4.3.1 How to Install

To install our framework, one needs to make sure the NeOn Toolkit 1.2.2 has been installed. Then start NeOn Toolkit and follow the steps below:

1. In NeOn Toolkit, go to “Help" → “Software Updates" → “Find and Install...".
2. Choose “Search for new features to install" and then click button “Next >".
3. Choose “NeOn Toolkit Update Site" and then click button "Finish".
4. Expand the list of “NeOn Toolkit Update Site" and then “diagnose and repair", afterwards choose “RaDON Changes Feature" and click button “Next >".
5. Choose “I accept the terms in the license agreement" and then click button “Next >".
6. Click “Finish".
7. In the wizard of “Feature Verification", choose “Install All".
8. Restart NeOn Toolkit.

4.3.2 How to Use

To annotate an axiom or show all relevant issues about the axiom in a justification found by RaDON, please follow the steps below:

1. Start Oyster server by clicking the button of “Oyster Registry" in the tool bar in NeOn Toolkit. Please note that, if you cannot start Oyster server, maybe you could check the following things:
   a) Check whether the port 1099 has been used in your machine or not. If this port has been used, you need to kill it and then start oyster.
   b) If the suggestion above can not resolve the problem, you could try to start Oyster by reading the ontologies locally. To do so, please go to the menu "Window" –> "Preferences" –> Oyster Storage Preference. In this configuration page, you could choose "Read ontologies locally".
   c) If you still get the problem, please try to delete the folders of "O2ServerFiles" and "server" in the root directory of NeOnToolkit and then copy the folder of "O2ServerFiles" which is newly downloaded to this directory.
2. Right-click an ontology O in the ontology navigator and choose “log changes" (Please make sure that you have logged in already. Otherwise, go to “Window" → “Preferences..." and log in in the tab of “Collaborative Development Preference". Before login, Oyster server should have been launched.).
3. In O's “Entity properties" page, input the Cicero project URL in the tab of “Argumentation Settings".
4. Right-click O in the ontology navigator and choose “Diagnose and Repair...".

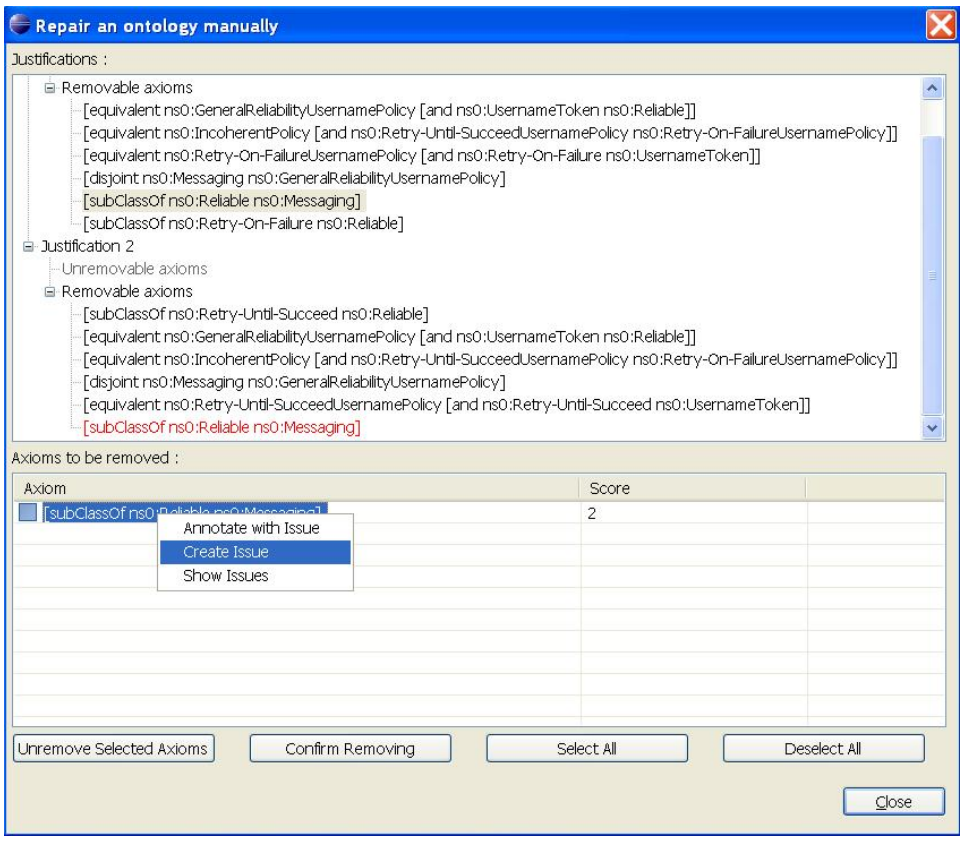

Figure 4.2: Annotate axioms in RaDON using Cicero

5. If O is incoherent, click the button of “Handle Incoherence" and then click ‘+’ before an unsatisfiable concept to compute its justifications. After this, choose “Repair Manually".

6. Choose some axioms in the section of “Justifications" to be removed to resolve the incoherence. Then one could annotate the axioms in the section of “Axioms to be removed" by right-clicking an axiom here and choosing an option regarding annotation (see Figure 4.2).

Once some axioms have been moved from an ontology to resolve the incoherence (or inconsistency), one could annotate the changes or show all relevant issues about an change by the following steps:

1. The same steps as steps 1 - 5 above.

2. Confirm the removal. That is, one could choose some axioms in the section of “Axioms to be removed" and then click the button “Confirm Removal". In such case, the chosen axioms will be removed from the ontology.

3. Open the workflow plug-in by choosing “Window" → “Show view" → “Other..." → “Workflow" → “Draft View".

4. In the “Draft View", one could see the changes made in RaDON by clicking the button of “Refresh Changes List".

5. One could annotate an change or show all its relevant issues by right-clicking an change and choosing a corresponding action (see Figure 4.3).

The users could use the four workflow views (i.e. Draft View, To Be Approved View, Approved View and To Be Deleted View) to manage the changes made by RaDON. In each view, the users can annotate an

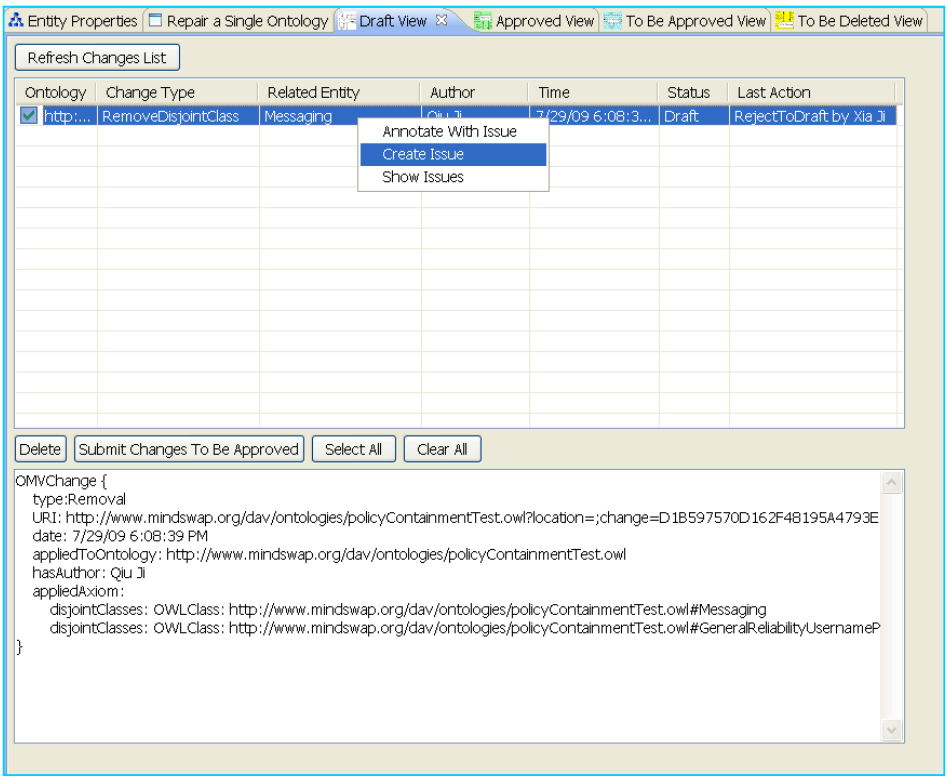

Figure 4.3: Annotate changes using Cicero

change with an issue or show all relevant issues. It is noted that, for different workflow view, different roles are needed to do some changes. For example, for the “Draft View", only “Subject Expert" can manage the changes here. For other views, “Validator" is needed. The users could change their roles in the tab of “Collaborative Development Preference" in menu “Preferences..." which is a submenu of “Window" in NeOn Toolkit.

Chapter 5

Contextual Reasoning in Possibilistic Description Logics

5.1 Motivation

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 D3.1.3, we introduced a model for representing provenance. In this model, 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. We then considered how provenance context can be exploited to deal with inconsistency and uncertainty (see Section 4.1.2 in D3.1.3). We also give explanations on how to use possibilistic OWL to represent and reason with provenance context (see Section 4.1.4 in D3.1.3).

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].

The application of possibilistic logic to deal with uncertainty in the Semantic Web is first studied in [Hol95] and is then discussed in [DMP06]. When we obtain an ontology using ontology learning techniques, the axioms of the ontology are often attached with confidence degrees and the learned ontology may be inconsistent [HV05]. In this case, possibilistic logic provides a flexible framework to interpret the confidence values and to reason with the inconsistent ontology under uncertainty. Another family of important approaches that extend description logics with uncertainty reasoning are probabilistic description logics [Jae94, Hei94, GL02, DS05, NF04, DP04, KLP97]. We list some major differences between possibilistic extension and probabilistic extension. First, possibilistic description logics are based on possibilistic semantics, thus the notion of consistency is in line with that in classical description logics. In contrast, probabilistic extensions are based on probabilistic measures, thus follow a different definition of consistency. Second, unlike probabilistic logic, the weight attached to an axiom in possibilistic logic is not absolute and can be be replaced by another number as long as the ordering between two weights is not changed. In summary, possibilistic description logics are more flexible than probabilistic ones to deal with inconsistency and qualitative uncertainty.

There exist problems which need further discussion for possibilistic description logics. First, there is no formal definition of the semantics of possibilistic description logics, although a proof method for possibilistic description logics is given. Second, there is no implementation of possibilistic inference given in possibilistic description logics.

In this chapter, we present a possibilistic extension of description logics. We first give the syntax and semantics of possibilistic logics. We then define some inference services in possibilistic description logics. Finally, we implement the algorithms for inference services in possibilistic description logics using KAON2 reasoner.

5.2 Preliminaries

5.2.1 Possibilistic Logic

Possibilistic logic [DLP94] is a weighted logic where each classical logic formula is associated with a number in $(0, 1]$. Semantically, the most basic and important notion is *possibility distribution* π: $\Omega \rightarrow [0, 1]$, where Ω is the set of all classical interpretations. $\pi(\omega)$ represents the degree of compatibility of interpretation ω with available beliefs. From *possibility distribution* π, two measures can be determined, one is the possibility degree of formula ϕ, defined as $\Pi(\phi) = max\{\pi(\omega) : \omega \models \phi\}$, the other is the necessity or certainty degree of formula ϕ, defined as $N(\phi) = 1 - \Pi(\neg\phi)$.

At syntactical level, a *possibilistic formula* is a pair (ϕ, α) consisting of a classical logic formula ϕ and a degree α expressing certainty or priority. A possibilistic knowledge base is the set of possibilistic formulas of the form $B = \{(\phi_i, \alpha_i) : i = 1, ..., n\}$. The classical base associated with B, denoted B^*, is defined as $B^* = \{\phi_i|(\phi_i, \alpha_i) \in B\}$. A possibilistic knowledge base is consistent iff its classical base is consistent.

Given a possibilistic knowledge base B and $\alpha\in(0, 1]$, the α-cut (strict α-cut) of B is $B_{\geq\alpha} = \{\phi\in B^*|(\phi, \beta)\in B \ and \ \beta\geq\alpha\}$ ($B_{>\alpha} = \{\phi\in B^*|(\phi, \beta)\in B \ and \ \beta>\alpha\}$). The *inconsistency degree* of B, denoted $Inc(B)$, is defined as $Inc(B) = max\{\alpha_i : B_{\geq\alpha_i} \ is \ inconsistent\}$.

There are two possible definitions of inference in possibilistic logic.

Definition 1 *Let B be a possibilistic knowledge base.*

- *A formula ϕ is said to be a* plausible consequence *of B, denoted by $B\vdash_P\phi$, iff $B_{>Inc(B)} \vdash \phi$.*
- *A formula ϕ is said to be a* possibilistic consequence *of B to degree α, denoted by $B\vdash_\pi(\phi, \alpha)$, iff the following conditions hold: (1) $B_{\geq\alpha}$ is consistent, (2) $B_{\geq\alpha}\vdash\phi$, (3) $\forall\beta>\alpha$, $B_{\geq\beta} \nvdash \phi$.*

According to Definition 1, an inconsistent possibilistic knowledge base can non-trivially infer conclusion, so it is inconsistency tolerent. However, it suffers from the “drowning problem" [BCD$^+$93]. That is, given an inconsistent possibilistic knowledge base B, formulas whose certainty degrees are not larger than $Inc(B)$ are completely useless for nontrivial deductions. For instance, let $B = \{(p, 0.9), (\neg p, 0.8), (r, 0.6), (q, 0.7)\}$, it is clear that B is equivalent to $B = \{(p, 0.9), (\neg p, 0.8)\}$ because $Inc(B) = 0.8$. So $(q, 0.7)$ and $(r, 0.6)$ are not used in the possibilistic inference.

Several variants of possibilistic inference have been proposed to avoid the drowning effect. One of them, called linear order inference, is defined as follows.

Definition 2 *Let $B = \{(\phi_i, \alpha_i) : i = 1, ..., n\}$ be a possibilistic knowledge base. Suppose β_j ($j = 1, ..., k$) are all distinct weights appearing in B such that $\beta_1 > \beta_2 > ... > \beta_k$. Let $\Sigma_B = (S_1, ..., S_k)$, where $S_i = \{\phi_l : (\phi_l, \alpha_l)\in B, \alpha_l = \beta_i\}$, and $\Sigma_{LO,B} = \bigcup_{i=1}^{k} S_i'$, where S_i' is defined by $S_i' = S_i$ if $S_i \cup \bigcup_{j=1}^{i-1} S_j'$ is consistent, $\emptyset$ otherwise. A formula ϕ is said to be a linear consequence of B, denoted $B \vdash_{LO} \phi$, iff $\Sigma_{LO,B} \vdash \phi$.*

The linear order approach does not stop at the inconsistency degree of possibilistic knowledge base B. It takes into account of formulas whose certainty degrees are less than the inconsistency degree.

5.3 Possibilistic Description Logics

5.3.1 Syntax

The syntax of possibilistic DL is based on the syntax of classical DL. A *possibilistic axiom* is a pair (ϕ, α) consisting of an axiom ϕ and a weight $\alpha \in (0, 1]$. A *possibilistic TBox* (resp., *ABox*) is a finite set of possibilistic axioms (ϕ, α), where ϕ is an TBox (resp., ABox) axoim. A possibilistic DL knowledge base $\mathcal{B} = (\mathcal{T}, \mathcal{A})$ consists of a possibilistic TBox $\mathcal{T}$ and a possibilistic ABox $\mathcal{A}$. We use $\mathcal{T}^*$ to denote the classical DL axioms associated with $\mathcal{T}$, i.e., $\mathcal{T}^* = \{\phi_i : (\phi_i, \alpha_i) \in \mathcal{T}\}$ ($\mathcal{A}^*$ can be defined similarly). The classical base $\mathcal{B}^*$ of a possibilistic DL knowledge base is $\mathcal{B}^* = (\mathcal{T}^*, \mathcal{A}^*)$. A possibilistic DL knowledge base $\mathcal{B}$ is inconsistent if and only if $\mathcal{B}^*$ is inconsistent.

Given a possibilistic DL knowledge base $\mathcal{B} = (\mathcal{T}, \mathcal{A})$ and $\alpha \in (0, 1]$, the α-cut of $\mathcal{T}$ is $\mathcal{T}_{\geq\alpha} = \{\phi \in \mathcal{B}^* | (\phi, \beta) \in \mathcal{T} \ and \ \beta \geq \alpha\}$ (the α-cut of $\mathcal{A}$, denoted as $\mathcal{A}_{\geq\alpha}$, can be defined similarly). The strict α-cut of $\mathcal{T}$ (resp., $\mathcal{A}$) can be defined similarly as the strict cut in possibilistic logic. The α-cut (resp., strict α-cut) of $\mathcal{B}$ is $\mathcal{B}_{\geq\alpha} = (\mathcal{T}_{\geq\alpha}, \mathcal{A}_{\geq\alpha})$ (resp., $\mathcal{B}_{>\alpha} = (\mathcal{T}_{>\alpha}, \mathcal{A}_{>\alpha})$). The *inconsistency degree* of $\mathcal{B}$, denoted $Inc(\mathcal{B})$, is defined as $Inc(\mathcal{B}) = max\{\alpha_i : \mathcal{B}_{\geq\alpha_i} \ is \ inconsistent\}$.

We use the following example as a running example throughout this paper.

Example 1 *Suppose we have a possibilistic DL knowledge base* $\mathcal{B} = (\mathcal{T}, \mathcal{A})$, *where* $\mathcal{T} = \{(Eat_{fish} \sqsubseteq Swim, 0.6), (Bird \sqsubseteq Fly, 0.8), (HasWing \sqsubseteq Bird, \ 0.95)\}$ *and* $\mathcal{A} = \{(Bird(chirpy), 1), (HasWing(tweety), 1), (\neg Fly(tweety), 1)\}$. *The TBox* $\mathcal{T}$ *states that it is rather certain that birds can fly and it is almost certain that something with wing is a bird. The ABox* $\mathcal{A}$ *states that it is certain that tweety has wing and it cannot fly, and chirpy is a bird. Let* $\alpha = 0.8$. *We then have* $\mathcal{B}_{\geq 0.8} = (\mathcal{T}_{\geq 0.8}, \mathcal{A}_{\geq 0.8})$, *where* $\mathcal{T}_{\geq 0.8} = \{Bird \sqsubseteq Fly, HasWing \sqsubseteq Bird\}$ *and* $\mathcal{A}_{\geq 0.8} = \{HasWing(tweety), \neg Fly(tweety), Bird(chirpy)\}$. *It is clear that* $\mathcal{B}_{\geq\alpha}$ *is inconsistent. Now let* $\alpha = 0.95$. *Then* $\mathcal{B}_{\geq\alpha} = (\mathcal{T}_{\geq 0.95}, \mathcal{A}_{\geq 0.95})$, *where* $\mathcal{T}_{\geq 0.95} = \{HasWing \sqsubseteq Bird\}$ *and* $\mathcal{A}_{\geq 0.95} = \{HasWing(tweety), \neg Fly(tweety), Bird(chirpy)\}$. *So* $\mathcal{B}_{\geq\alpha}$ *is consistent. Therefore,* $Inc(\mathcal{B}) = 0.8$.

5.3.2 Semantics

The semantics of possibilistic DL is defined by a *possibility distribution* π over the set $\mathbf{I}$ of all classical description logic interpretations, i.e., $\pi : \mathbf{I} \rightarrow [0, 1]$. $\pi(I)$ represents the degree of compatibility of interpretation I with available information. For two interpretations I_1 and I_2, $\pi(I_1) > \pi(I_2)$ means that I_1 is preferred to I_2 according to the available information. Given a possibility distribution π, we can define the possibility measure Π and necessity measure N as follows: $\Pi(\phi) = max\{\pi(I) : I \in \mathbf{I}, I \models \phi\}$ and $N(\phi) = 1 - max\{\pi(I) : I \not\models \phi\}$[1]. Unlike possibilistic logic, the necessary measure cannot be not defined by the possibility measure because the negation of an axiom is not defined in traditional DLs. However, given a DL axiom ϕ, let us define the negation of ϕ as $\neg\phi = \exists(C \sqcap \neg D)$ if $\phi = C \sqsubseteq D$ and $\neg\phi = \neg C(a)$ if $\phi = C(a)$, where $\exists(C \sqcap \neg D)$ is an existence axiom (see the discussion of negation of a DL axiom in [FHP+06]), then it is easy to check that $N(\phi) = 1 - \Pi(\neg\phi)$. Given two possibility distributions π and π', we say that π is more specific (or more informative) than π' iff $\pi(I) \leq \pi'(I)$ for all $I \in \Omega$. A possibility distribution π satisfies a possibilistic axiom (ϕ, α), denoted $\pi \models (\phi, \alpha)$, iff $N(\phi) \geq \alpha$. It satisfies a possibilistic DL knowledge base $\mathcal{B}$, denoted $\pi \models \mathcal{B}$, iff it satisfies all the possibilistic axioms in $\mathcal{B}$.

Given a possibilistic DL knowledge base $\mathcal{B} = \langle \mathcal{T}, \mathcal{A} \rangle$, we can define a possibility distribution from it as follows: for all $I \in \mathbf{I}$,

$$\pi_{\mathcal{B}}(I) = \begin{cases} 1 & \text{if } \forall \phi_i \in \mathcal{T}^* \cup \mathcal{A}^*, I \models \phi_i, \\ 1 - max\{\alpha_i | I \not\models \phi_i, (\phi_i, \alpha_i) \in \mathcal{T} \cup \mathcal{A}\} & \text{otherwise.} \end{cases} \tag{5.1}$$

As in possibilistic logic, we can also show that the possibility distribution defined by Equation 5.1 is the least specific possibility distribution satisfying $\mathcal{B}$. Let us consider Example 1 again. $I = \langle \Delta^I, \cdot^I \rangle$ is an

[1] The definition of necessity measure is pointed out by one of the reviewers.

interpretation, where $\Delta^I = \{tweety, chirpy\}$ and $Bird^I = \{tweety, chirpy\}$, $Fly^I = \{chirpy\}$, and $HasWing^I = \{tweety\}$. It is clear that I satisfies all the axioms except $Bird \sqsubseteq Fly$ (whose weight is 0.8), so $\pi_{\mathcal{B}}(I) = 0.2$.
Let us give some properties of the possibility distribution defined by Equation (5.1).

Theorem 1 *Let $\mathcal{B}$ be a possibilistic DL knowledge base and $\pi_{\mathcal{B}}$ be the possibility distribution obtained by Equation (5.1). Then $\mathcal{B}$ is consistent if and only if there exists an interpretation $\mathcal{I}$ such that $\pi_{\mathcal{B}}(\mathcal{I}) = 1$.*

Proposition 1 *Let $\mathcal{B}$ be a possibilistic DL knowledge base and $\pi_{\mathcal{B}}$ be the possibility distribution obtained by Equation 5.1. Then $Inc(\mathcal{B}) = 1 - max_{I \in \mathbf{I}} \pi_{\mathcal{B}}(I)$.*

5.3.3 Possibilistic inference in possibilistic DLs

We consider the following inference services in possibilistic DLs.

- Instance checking: an individual a is a *plausible* instance of a concept C with respect to a possibilistic DL knowledge base $\mathcal{B}$, written $\mathcal{B} \models_P C(a)$, if $\mathcal{B}_{>Inc(\mathcal{B})} \models C(a)$.
- Instance checking with necessity degree: an individual a is an instance of a concept C to degree α with respect to $\mathcal{B}$, written $\mathcal{B} \models_\pi (C(a), \alpha)$, if the following conditions hold: (1) $\mathcal{B}_{\geq\alpha}$ is consistent, (2) $\mathcal{B}_{\geq\alpha} \models C(a)$, (3) for all $\beta > \alpha$, $\mathcal{B}_{\geq\beta} \not\models C(a)$.
- Instance checking with necessity degree: an individual a is an instance of a concept C to degree α with respect to $\mathcal{B}$, written $\mathcal{B} \models_\pi (C(a), \alpha)$, if the following conditions hold: (1) $\mathcal{B}_{\geq\alpha}$ is consistent, (2) $\mathcal{B}_{\geq\alpha} \models C(a)$, (3) for all $\beta > \alpha$, $\mathcal{B}_{\geq\beta} \not\models C(a)$.
- Subsumption with necessity degree: a concept C is subsumed by a concept D to a degree α with respect to a possibilistic DL knowledge base $\mathcal{B}$, written $\mathcal{B} \models_\pi (C \sqsubseteq D, \alpha)$, if the following conditions hold: (1) $\mathcal{B}_{\geq\alpha}$ is consistent, (2) $\mathcal{B}_{\geq\alpha} \models C \sqsubseteq D$, (3) for all $\beta > \alpha$, $\mathcal{B}_{\geq\beta} \not\models C \sqsubseteq D$.

We illustrate the inference services by reconsidering Example 1.

Example 2 *(Example 1 continued) According to Example 1, we have $Inc(\mathcal{B}) = 0.8$ and $\mathcal{B}_{>0.8} = (\mathcal{T}_{>0.8}, \mathcal{A}_{>0.8})$, where $\mathcal{T}_{>0.8} = \{HasWing \sqsubseteq Bird\}$ and $\mathcal{A}_{>0.8} = \{HasWing(tweety), \neg Fly(tweety), Bird(chirpy)\}$. Since $\mathcal{B}_{>0.8} \models Bird\ (tweety)$, we can infer that $tweety$ is plausible to be a bird from $\mathcal{B}$. Furthermore, since $\mathcal{B}_{\geq 0.95} \models Bird(tweety)$ and $\mathcal{B}_{\geq 1} \not\models Bird(tweety)$, we have $\mathcal{B} \models_\pi (Bird(tweety), 0.95)$. That is, we are almost certain that tweety is a bird.*

5.3.4 Linear order inference in possibilistic DLs

Possibilistic inference in possibilistic DL inherits the drowning effect of possibilistic inference in possibilistic logic. We adapt and generalize the linear order inference to deal with the drowning problem.

Definition 3 *Let $\mathcal{B} = (\mathcal{T}, \mathcal{A})$ be a possibilistic DL knowledge base. Suppose β_j ($j = 1, ..., k$) are all distinct weights appearing in $\mathcal{B}$ such that $\beta_1 > \beta_2 > ... > \beta_k$. Let $\mathcal{B}' = \cup\mathcal{T} \cup \mathcal{A}$. Let $\Sigma_{\mathcal{B}} = (\mathcal{S}_1, ..., \mathcal{S}_k)$, where $\mathcal{S}_i = \{(\phi_l, \alpha_l) : (\phi_l, \alpha_l) \in \mathcal{B}', \alpha_l = \beta_i\}$, and $\Sigma_{LO,\mathcal{B}} = \bigcup_{i=1}^{k} \mathcal{S}_i'$, where $\mathcal{S}_i'$ is defined by $\mathcal{S}_i' = \mathcal{S}_i$ if $\mathcal{S}_i \cup \bigcup_{j=1}^{i-1} \mathcal{S}_j'$ is consistent, $\emptyset$ otherwise. Let ϕ be a query of the form $C(a)$ or $C \sqsubseteq D$. Then*

- *ϕ is said to be a consequence of $\mathcal{B}$ $w.r.t$ the linear order policy, denoted $\mathcal{B} \vdash_{LO} \phi$, iff $(\Sigma_{LO,\mathcal{B}})^* \vdash \phi$.*
- *ϕ is said to be a weighted consequence of $\mathcal{B}$ to a degree α $w.r.t$ the linear order policy, denoted $\mathcal{B} \vdash_{LO} (\phi, \alpha)$, iff $\Sigma_{LO,\mathcal{B}} \vdash_\pi (\phi, \alpha)$.*

Algorithm 1: Compute the inconsistency degree

Data: $\mathcal{B} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T} \cup \mathcal{A} = \{(\phi_i, \alpha_i) : \alpha_i \in (0,1], i = 1, ..., n\}$, where n is the number of axioms in the testing ontology $\mathcal{B}$;

Result: The inconsistency degree d

```
begin
    b := 0          // b is the begin pointer of the binary search
    m := 0          // m is the middle pointer of the binary search
    d := 0.0        // The initial value of inconsistency degree d is set to be 0.0
    W = Asc(α_1, ..., α_n)
    W(−1) = 0.0 // The special element −1 of W is set to be 0.0
    e := |W| − 1  // e is the end pointer of the binary search
    if B_{≥W(0)} is consistent then
        d:=0.0
    else
        while b ≤ e do
            if b = e then
                return b
            m := ⌜(b + e)/2⌝
            if B_{≥W(m)} is consistent then
                e := m − 1
            else
                b := m + 1
        d := W(b)
end
```

In Definition 3, we not only define the consequence of a possibilistic DL knowledge base $w.r.t$ the linear order policy, but also the weighted consequence of it. The weighted consequence of $\mathcal{B}$ is based on the possibilistic inference.

Example 3 *(Example 1 continued) Let* $\phi = Eat_{fish} \sqsubseteq Swim$. *According to Example 2,* ϕ *is not a consequence of* $\mathcal{B}$ $w.r.t.$ *the possibilistic inference. Since* $\Sigma_{\mathcal{B}} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4)$, *where* $\mathcal{S}_1 = \mathcal{A}$, $\mathcal{S}_2 = \{(HasWing \sqsubseteq Bird, 0.95)\}$, $\mathcal{S}_3 = \{(Bird \sqsubseteq Fly, 0.8)\}$ *and* $\mathcal{S}_4 = \{(Eat_{fish} \sqsubseteq Swim, 0.6)\}$, *we have* $\Sigma_{LO,\mathcal{B}} = S_1 \cup S_2 \cup S_4$. *It is easy to check that* $\mathcal{B} \vdash_{LO} (Eat_{fish} \sqsubseteq Swim, 0.6)$.

5.4 Algorithms for Inference in Possibilistic DLs

We give algorithms for the inference in possibilistic DLs.

Algorithm 1 computes the inconsistency degree of a possibilistic DL knowledge base using a binary search. The function Asc takes a finite set of numbers in $(0, 1]$ as input and returns a vector which contains those distinct numbers in the set in an ascending order. For example, $Asc(0.2, 0.3, 0.3, 0.1) = (0.1, 0.2, 0.3)$. Let $W = (\beta_1, ..., \beta_n)$ is a vector consisting of n distinct numbers, then $W(i)$ denotes β_i. If the returned inconsistency degree is 0, that is $W(-1) = 0$, it shows the ontology to be queried is consistent.

Since Algorithm 1 is based on binary search, to compute the inconsistency degree, it is easy to check that the algorithm requires at most $\ulcorner log_2 n \urcorner$+1 satisfiability check using a DL reasoner.

Algorithm 2 returns the necessity degree of an axiom inferred from a possibilistic DL knowledge base $w.r.t$ the possibilistic inference. We compute the inconsistency degree of the input ontology. If the axiom is a plausible consequence of a possibilistic DL knowledge base, then we compute its necessity degree using a binary search (see the first “if" condition). Otherwise, its necessity degree is 0, i.e., the default value given to w. Note that our algorithm is different from the algorithm given in [Lan00] for computing the necessity of

Algorithm 2: Possibilistic inference with certainty degrees

Data: $\mathcal{B} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T} \cup \mathcal{A} = \{(\phi_i, \alpha_i) : \alpha_i \in (0, 1], i = 1, ..., n\}$; a DL axiom ϕ.
Result: The certainty degree w associated with a query ϕ

```
begin
    m := 0
    w := 0.0 // The initial certainty degree of φ is set to be 0.0
    W = Asc(α_1, ..., α_n)
    W(−1) = 0.0
    e := |W| − 1
    compute l such that W(l) = Inc(B) //Inc(B) is computed by Algorithm 1
    b := l + 1
    if B_{≥W(b)} ⊨ φ then
        while b ≤ e do
            if b = e then
                return b
            m := ⌈(b + e)/2⌉
            if B_{≥W(m)} ⊭ φ then
                e := m − 1
            else
                b := m + 1
        w := W(b)
end
```

a formula in possibilistic logic (this algorithm needs to compute the negation of a formula, which is computationally hard in DLs according to [FHP$^+$06]). We consider only subsumption checking here. However, the algorithm can be easily extended to reduce instance checking as well.

In Algorithm 3, we call Algorithm 1 and Algorithm 2 to compute the certainty degree of the query ϕ $w.r.t$ the linear order inference. In the “while" loop, the first “if" condition checks if the inconsistency degree is greater than 0 and then delete the axioms whose necessity degrees are equal to the inconsistency degree. After that, we call Algorithm 1 to compute the inconsistency degree of the initial knowledge base or knowledge base obtained from the first “if" loop. Then the second “if" condition checks if the axiom is a plausible consequence of the possibilistic DL knowledge base and end the “while" loop if the answer is positive. The final “if" condition simply tests if the possibilistic DL knowledge base is consistent or not and terminate the “while" loop if the answer is positive. Finally, we compute the certainty degree of ϕ by calling Algorithm 2. This algorithm need to call polynomial times of satisfiability check using a DL reasoner.

Algorithms 2 and 3 compute inference with certainty degree because it is more difficult to obtain the certainty degree of an inferred axiom. They can be easily revised to compute plausible consequence. Because of the page limit, we do not provide the details here.

Proposition 2 *Let $\mathcal{B}$ be a possibilistic DL knowledge base and ϕ be a DL axiom. Deciding whether $\mathcal{B} \models_P \phi$ requires $\lceil log_2 n \rceil$+1 satisfiability check using a DL reasoner, where n is the number of distinct certainty degrees in $\mathcal{B}$. Furthermore, deciding whether $\mathcal{B} \models_\pi (\phi, \alpha)$ requires at most $\lceil log_2 n \rceil$+$\lceil log_2 n - l \rceil$+1 satisfiability check using a DL reasoner, where where n is the number of distinct certainty degrees in $\mathcal{B}$ and l is the inconsistency degree of $\mathcal{B}$.*

5.5 Reasoning With Distributed Systems in Possibilistic DL

In this section, we show how we can use possibilistic DL to reason with distributed systems which consists of ontologies and the mappings between them.

Algorithm 3: Linear order inference with certainty degrees

Data: $\mathcal{B} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T} \cup \mathcal{A} = \{(\phi_i, \alpha_i) : \alpha_i \in (0, 1], i = 1, ..., n\}$; a DL axiom ϕ.
Result: The certainty degree w associated with a query ϕ

```
begin
    d := 0.0 // The initial inconsistency degree is set to be 0.0
    w := 0.0 // The initial certainty degree of φ is set to be 0.0
    hasAnswer := false
    W = Asc(α_1, ..., α_n)
    e := |W| − 1 // e is a global variable to pass values to the subroutines
    while !hasAnswer do
        if d > 0 then
            e := d − 1
            B := B \ B_{=d}
            W := W \ d
        d := alg1(B), where alg1 is Algorithm 1
        if B_{>d} ⊨ φ then
            hasAnswer := true
        if d ≤ 0 then
            break
    if hasAnswer then
        w := alg2(B, φ), where alg2 is Algorithm 2
end
```

We first introduce the notion of correspondence between elements of two ontologies and mapping defined in [ES07]. Note that our definitions are compatible with the mapping metamodel given in NeOn deliverable D1.1.2 [HBP+07a].

Given two ontologies O_1 and O_2, describing the same or largely overlapping domains of interest. We can define the correspondences between elements of them.

Definition 4 *Let O_1 and O_2 be two ontologies, Q be a function that defines sets of matchable elements $Q(O_1)$ and $Q(O_2)$. A correspondence is a 4-tuple $\langle e, e', r, \alpha \rangle$ such that $e \in Q(O_1)$ and $e' \in Q(O_2)$, r is a semantic relation, and α is a confidence value from a suitable structure $\langle D, \leq \rangle$.*

In Definition 4, there is no restriction on function Q, semantic relation r and domain D. Similar to the paper [MST07], we only consider correspondences between concepts and restrict r to be one of the semantic relations from the set $\{\equiv, \sqsubseteq, \sqsupseteq\}$, and assume $D = [0.0, 1.0]$.

From a set of correspondences, we can define the notion of a mapping as follows. We follow the definition of a mapping given in [HBP+07a].

Definition 5 *Given ontologies O_1 and O_2, let Q be a function that is given in Definition 4. $\mathcal{M}$ is a set of correspondences. Then $\mathcal{M}$ is a mapping between O_1 and O_2 iff for all correspondences $\langle e, e', r, \alpha \rangle \in \mathcal{M}$ we have $e \in Q(O_1)$ and $e' \in Q(O_2)$.*

That is, a mapping is a set of correspondences whose elements are matchable.

Definition 6 *[ZE06] A distributed system is a triple $\mathcal{D} = \langle (O_i), (A_{ij}) \rangle$, consisting of a family of ontologies $(O_i)_{i \in I}$ over a set of indexes I interconnected by a family of alignments $(A_{ij})_{i,j \in I}$.*

Example 4 *Take the two ontologies CRS and EKAW in the domain of conference management systems as an example. They contain the following axioms:*

$$\begin{array}{ll} \text{crs : article} \sqsubseteq \text{crs : document}, & \text{crs : program} \sqsubseteq \neg\text{crs : document}, \\ \text{ekaw : Paper} \sqsubseteq \text{ekaw : Document}, & \text{ekaw : Workshop\_Paper} \sqsubseteq \text{ekaw : Paper} \\ \text{ekaw : Conference\_Paper} \sqsubseteq \text{ekaw : Paper}, & \text{ekaw : PC\_Member} \sqsubseteq \text{ekaw : Possible\_Reviewer}, \end{array}$$

We also assume that EKAW contains assertional axioms ekaw:Workshop_Paper(paper1), ekaw:Workshop_Paper(paper2). The correspondences in the mapping A_{12} between O_1 and O_2 are listed as follows:

$$\begin{array}{lll} \text{m}_1 : \langle \text{crs : article, ekaw : Conference\_Paper}, & \sqsubseteq, & 0.65\rangle \\ \text{m}_2 : \langle \text{ekaw : Workshop\_Paper, crs : article}, & \sqsubseteq, & 0.65\rangle \\ \text{m}_3 : \langle \text{ekaw : Document, crs : program}, & \sqsubseteq, & 0.80\rangle \\ \text{m}_4 : \langle \text{crs : program, ekaw : Document}, & \sqsubseteq, & 0.80\rangle \\ \text{m}_5 : \langle \text{crs : document, ekaw : Document}, & \sqsubseteq, & 0.93\rangle \end{array}$$

Definition 7 *Let $\mathcal{D} = \langle (O_i), (A_{ij})\rangle$ be a distributed system. The* union $\cup(\mathcal{D})$ *of $(O_i)_{i\in I}$ and $(A_{ij})_{i,j\in I}$ is defined as $\cup(\mathcal{D}) = \cup_{i\in I}\{(\phi, 1) : \phi \in O_i\} \cup \cup_{i,j\in I}\{(t(m), \alpha) : m \in A_{ij}\}$ with t being a translation function that converts a correspondence into an axiom in the following way: $t(\langle C, C', r, \alpha\rangle) = CrC'$.*

That is, we first translate all the correspondences in all the mapping A_{ij} into DL axioms, then the union of the ontologies connected and mappings is the set-union of axioms in ontologies attached with weight 1 and the translated axioms attached with the corresponding weights. This union is a possibilistic DL knowledge base. Take a correspondence in Example 4 as an example, we have t($\langle$crs:article, ekaw:Conference_Paper, $\sqsubseteq$, 0.65$\rangle$) = crs:article $\sqsubseteq$ ekaw:Conference_Paper. We have $\cup(\mathcal{D})$={(ekaw:Workshop_Paper(paper1),1), (ekaw:Workshop_Paper(paper2),1), (crs:article $\sqsubseteq$ crs:document,1), (crs: program $\sqsubseteq$ $\neg$ crs:document,1), (ekaw:Paper $\sqsubseteq$ ekaw:Document,1), (ekaw:Workshop_Paper $\sqsubseteq$ ekaw:Paper,1), (ekaw:Conference_Paper $\sqsubseteq$ ekaw:Paper,1), (ekaw:PC_Member $\sqsubseteq$ ekaw:Possible_Reviewer,1), (crs:article $\sqsubseteq$ ekaw:Conference_Paper,0.65), (ekaw:Workshop_Paper $\sqsubseteq$ crs:article,0.65), (ekaw:Document $\sqsubseteq$ crs:program,0.80), (crs:program $\sqsubseteq$ ekaw:Document,0.80), (crs:document $\sqsubseteq$ ekaw:Document,0.93)}. The inconsistency degree of $\cup(\mathcal{D})$ is 0.80. By applying possibilistic inference, we can infer that crs:document is subsumed by ekaw:Document with degree 0.93.

5.6 Implementation

We have implemented all the algorithms about inference in possibilistic DL as a plug-in in NeOn Toolkit. This plug-in can be used to compute the inconsistency degree for an inconsistent ontology. It also provides the functionalities to do possibilistic inference or linear order inference with certainty degree. In order to handle mapped ontologies, the users are required to merge the ontologies first, then use our system to do inference. Note that our system can only deal with mappings consisting of correspondences attached with different weights.

To install the possibilistic DL plug-in into NeOn Toolkit, the users could follow the similar steps as those for installing the framework in Section 4.3 except Step 4. That is, instead of choosing “RaDON Changes Feature", we should select “Possibilistic DL Feature". After installing possibilistic DL plug-in, the users could right-click an OWL ontology O in the ontology navigator in NeOn Toolkit and then choose the option of “Possibilistic Instance / Subsumption Checking" to show the view. Some information about O (e.g. the number of axioms in the ontology) will be shown in the section of “Ontology Details" in this view. Before we perform any actions in this view, a weight file associated to O needs to be located in the “Weight" tab in “Ontology Details" section.

In order to compute the inconsistency degree, the users could go to the tab of “Configuration" and choose the first option. Usually, the first option is the default one. By clicking the button of “Check Inconsistency" in the section of “Compute Inconsistency Degree", we can get the answer about whether this ontology is inconsistent or not with the corresponding inconsistency degree. See Figure 5.1 as an example. The ontology is inconsistent and its inconsistency degree is 0.122.

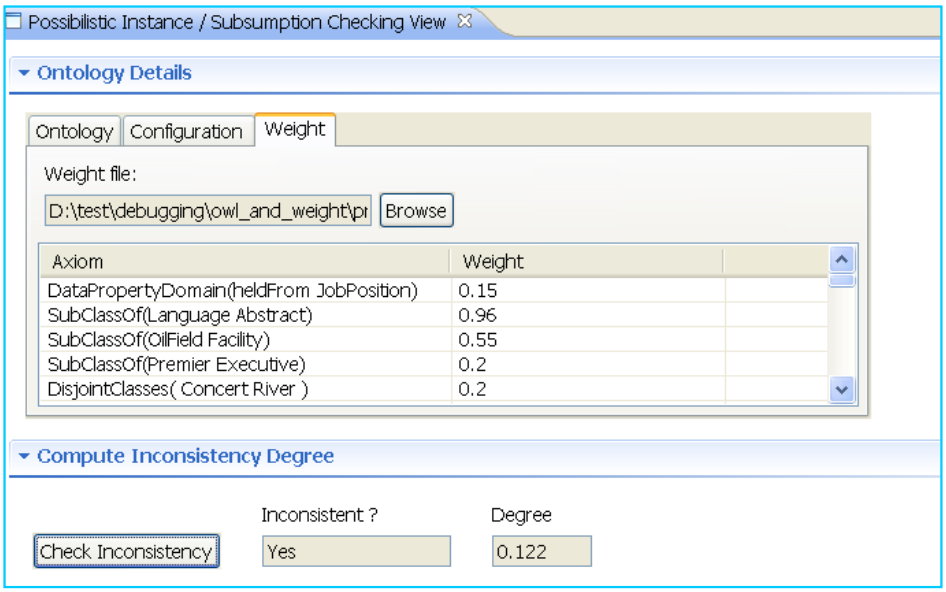

Figure 5.1: Compute inconsistency degree

As for possibilistic instance or subsumption checking, we have two different methods: possibilistic inference and linear order inference. The users could choose different methods in the tab of “Configuration" in the Section of “Ontology Details" (see Figure 5.2). After choosing a specific method, the corresponding section will be expanded automatically right below the section of “Ontology Details". In the newly expanded section, the users could do instance checking or subsumption checking. For example, the users could select one individual and one concept from the corresponding lists and then click the button of “Instance Checking". If the chosen instance can be inferred that it belongs to the chosen concept, then a weight between 0 and 1 will be shown. Otherwise, the weight is 0. In the example in Figure 5.2, we can infer the individual canal_3 is an individual of concept Canal with certainty degree 0.3 by applying the possibilistic inference.

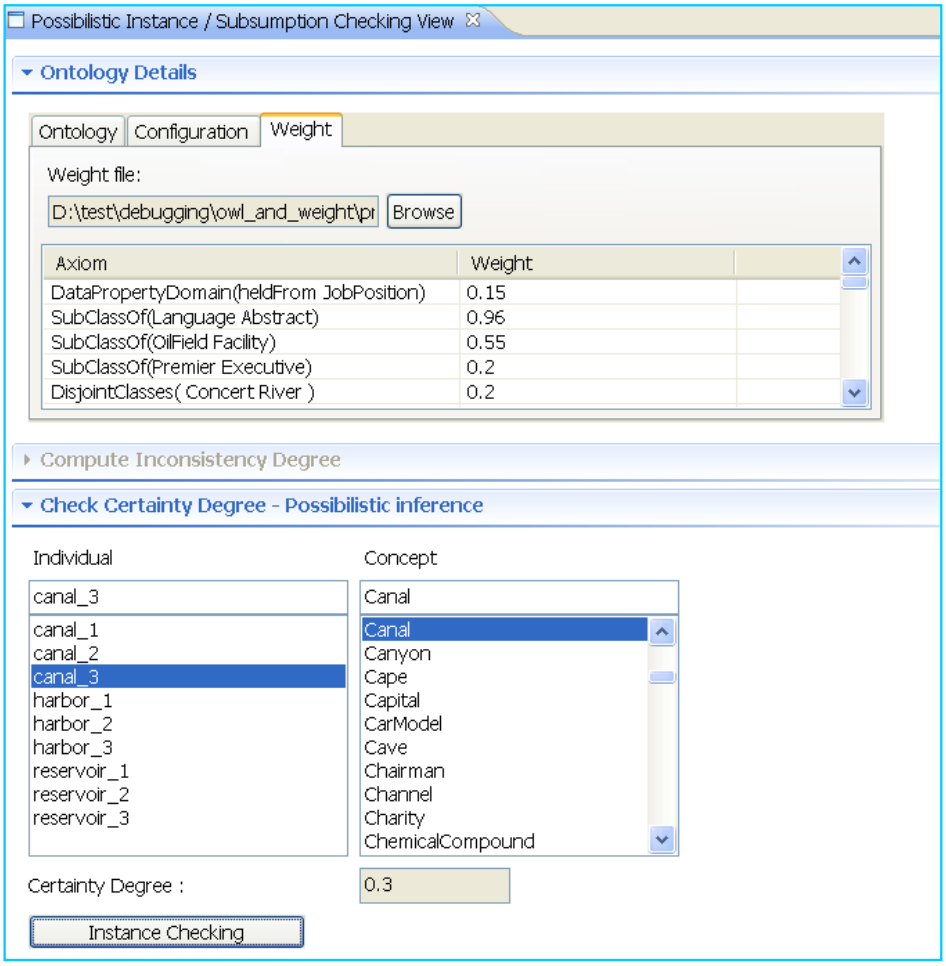

Figure 5.2: Check certainty degree

Chapter 6

Conclusion

6.1 Summary

In this deliverable, we proposed some methods for contextual reasoning. We first presented a contextual framework for diagnosis and repair. We implemented a system by integrating RaDON and Cicero through NeOn Toolkit and proposed an enhanced ontology editing framework. In our framework, we used the RaDON system to debug and diagnose inconsistencies and used the Cicero system to provide a discussion forum to the users.

We then presented possibilistic description logics which extend description logics to deal with uncertainty. We showed how possibilistic description logics can provide an integrated framework to reason with mappings and uncertainty. We gave semantics and syntax of possibilistic description logics. We then defined several inference services in possibilistic description logics, including a drowning-free variant of possibilistic inference, called linear order inference. We showed how some inference services in possibilistic description logics can be reduced to the problem of computing the *inconsistency degree* of the description logic-based knowledge base. We provided an algorithm to compute the inconsistency degree of a description logic-based and an algorithm for linear ordered inference. All these algorithm were implemented and a plugin has been developed for NeOn toolkit.

6.2 Roadmap

In our future work, we will consider contextual reasoning based on meta knowledge. Approaches for providing meta knowledge for query answers in relational databases and RDF repositories exist, that are based on algebraic operations. As query answering in description logics in general does not boil down to algebraic evaluation of tree shaped query models, these formalizations do not easily carry over. In this work, we will propose a formalization of meta knowledge which is still algebraic, but allows for reasoning with conflicting and incomplete meta knowledge. The relationship between the new formalism and the existing one in D3.1.3 will be specified. We mainly focus on reasoning with meta knowledge. In D3.2.3, we used pinpointing as a technique to deal with inconsistency. In this work, we will use pinpointing to come up with meta knowledge formulas for description logics, which then can be evaluated algebraically, and describe our prototypical implementation.

Bibliography

[BCD+93] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency Management and Prioritized Syntax-Based Entailment. In *Proceedings of IJCAI'93*, pages 640–647, 1993.

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

[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.

[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.

[DGG+08] Klaas Dellschaft, Aldo Gangemi, Jose Manuel Gomez, Holger Lewen, Valentina Presutti, and Margherita Sini. D2.3.1 Practical Methods to Support Collaborative Ontology Design. Technical Report D2.3.1, Universität Koblenz-Landau, April 2008.

[DLP94] Didier Dubois, Jérôme Lang, and Henri Prade. Possibilistic logic. In Handbook of logic in Aritificial Intelligence and Logic Programming*, Volume 3*, pages 439–513. Oxford University Press, 1994.

[DMP06] Didier Dubois, Jérôme Mengin, and Henri Prade. Possibilistic uncertainty and fuzzy features in description logic: A preliminary discussion. In *Capturing Intelligence: Fuzzy Logic and the Semantic WEb*, pages 101–113. Elsevier, 2006.

[DP91] Didier Dubois and Henri Prade. Epistemic entrenchment and possibilistic logic. *Artificial Intelligence*, 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.

[ES07] Jerome Euzenat and Pavel Shvaiko. *Ontology Matching*. Berlin; Heidelberg, Springer, 2007.

[FHP+06] Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, Negations and Changes in Ontologies. In *Proceeding of AAAI'06*, pages 1295–1300, 2006.

[GL02] Rosalba Giugno and Thomas Lukasiewicz. P-shoq(d): A probabilistic extension of shoq(d) for probabilistic ontologies in the semantic web. In *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, pages 86–97, Cosenza, Italy, Sep 2002.

[HBP+07a] 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.

[HBP+07b] Peter Haase, Saartje Brockmans, Raul Palma, Jérôme Euzenat, and Mathieu d'Aquin. D1.1.2 updated version of the networked ontology model. NeOn Project Deliverable D1.1.2, Universität Karlsruhe (TH), 2007.

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

[Hol95] Bernhard Hollunder. An alternative proof method for possibilistic logic and its application to terminological logics. *Int. J. Approx. Reasoning*, 12(2):85–109, 1995.

[HV05] Peter Haase and Johanna Völker. Ontology learning and reasoning - dealing with uncertainty and inconsistency. In *Proceeding of URSW'05*, pages 45–55, 2005.

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

[JQH08] Qiu Ji, Guilin Qi, and Peter Haase. A relevance-based algorithm for finding justifications of DL entailments. In *Technical report, University of Karlsruhe, http://www.aifb.uni-karlsruhe.de/WBS/gqi/papers/RelAlg.pdf*, 2008.

[KLP97] Daphne Koller, Alon Y. Levy, and Avi Pfeffer. P-classic: A tractable probablistic description logic. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI'97/IAAI'97)*, pages 390–397, Providence, Rhode Island, Jul 1997.

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding All Justifications of OWL DL Entailments. In *Proceeding of 6th International and 2nd Asian Semantic Web Conference (ISWC/ASWC)*, pages 267–280, 2007.

[KR70] Werner Kunz and Horst Rittel. Issues as elements of information systems. WP 131, Institute of Urban and Regional Development, University of California, Berkeley, 1970.

[Lan00] Jérôme Lang. Possibilistic logic: complexity and algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 179–220. 2000.

[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.

[PB88] Colin Potts and Glenn Bruns. Recording the reasons for design decisions. In *ICSE*, pages 418–427, 1988.

[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*, 2004.

[QHJV07] Guilin Qi, Peter Haase, Qiu Ji, and Johanna Völker. D1.2.2 consistency models for networked ontologies evaluation. NeOn Project Deliverable D1.2.2, Universität Karlsruhe, 2007.

[Rei87] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[RW73] Horst W. J. Rittel and Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):155–169, June 1973.

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceeding of 18th International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 355–362, 2003.

[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 *Proceedings of the 3rd European Semantic Web Conference (ESWC'06)*, pages 241–256, Budva, Montenegro, Jun 2005.

[ZE06] Antoine Zimmermann and Jérôme Euzenat. Three semantics for distributed systems and their relations with alignment composition. In *Proc. of ISWC'06*, pages 16–29, 2006.