NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D1.2.3 Diagnosing and repairing inconsistent networked ontologies

Deliverable Co-ordinator: **Guilin Qi**

Deliverable Co-ordinating Institution: **University of Karlsruhe**

Other Authors: **Peter Haase, Qiu Ji**

In this deliverable, we propose a framework for diagnosing and repairing inconsistent networked ontologies. We first present an algorithm for finding *minimal unsatisfiability-preserving sub-ontologies (MUPS)* of an ontology w.r.t an unsatisfiable concept. We then propose an algorithm for repairing mappings in networked ontologies. Both algorithms are important to diagnose and repair inconsistency in networked ontologies. Our experiment shows that our algorithm for finding MUPS is faster than the best algorithm for finding all the MUPS of an ontology w.r.t. an unsatisfiable concept. We also implemented our algorithm for mapping repair and evaluate its efficiency and meaningfulness of the repaired result.

Document Identifier:	NEON/2008/D1.2.3/v1.0	Date due:	August 31, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2008
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 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

Change Log

Version	Date	Amended by	Changes
0.1	01-07-2008	Guilin Qi	Create the deliverable
0.2	04-08-2008	Guilin Qi	Chapter 2, Chapter 3
0.3	10-08-2008	Guilin Qi	Chapter 1, Chapter 4
0.4	15-08-2008	Qiu Ji	Chapter 5
0.5	20-08-2008	Guilin Qi	Executive Summary and Conclusion
0.6	20-08-2008	Qiu Ji	add examples
0.7	21-08-2008	Guilin Qi and Peter Haase	Final write-up
0.8	21-08-2008	Guilin Qi	Final QA
1.0	25-09-2008	Peter Haase	Final Version

Executive Summary

Next generation semantic applications are characterized by a large number of ontologies, some of them constantly evolving. As the complexity of semantic applications increases, more and more knowledge are embedded in applications, typically drawn from a wide variety of sources. This new generation of applications thus likely rely on ontologies embedded in a network of already existing ontologies. Ontologies and metadata have to be kept up to date when application environments and users' needs change. One of the major challenges in managing these networked and dynamic ontologies is to handle potential inconsistencies in single ontologies, and inconsistencies introduced by integrating multiple distributed ontologies.

We have proposed a general approach for dealing with inconsistency and incoherence in networked ontologies in NeOn deliverable D1.2.1 [QHJ07] and provided evaluation results on our approach in NeOn deliverable D1.2.2 [QHJV07]. According to the evaluation results given in deliverable D1.2.2, although the approach can be used to handle real life ontology in reasonable time, efficiency is still a problem if we want to deal with large ontologies. We especially need efficient algorithm for finding *minimal unsatisfiability-preserving sub-ontologies (MUPS)* of an ontology w.r.t an unsatisfiable concept. Moreover, we do not focus on networked ontologies in previous work. This work continues the work done in previous deliverables and considers diagnosing and repairing networked ontologies.

In this deliverable, we first propose an algorithm for finding MUPS of an ontology w.r.t an unsatisfiable concept. We first define a relevance-based ordering on the MUPS, which allows us to associate a relevance degree with each of the MUPS to facilitate the comparison among them. Specifically, we use a syntactic selection function based on concept relevance, whose intuition is to select axioms that are closely connected to the unsatisfiable concept. We then present an algorithm to find a set of MUPS for an unsatisfiable concept. The algorithm incrementally selects subsets of the ontology using the selection function and finds a set of MUPS from these sub-ontologies for the concept. When computing MUPS from a sub-ontology, our algorithm allows for different strategies: It either computes a set of MUPS that satisfies some condition(s) or computes all MUPS. Our algorithm is based on a black-box approach, and thus it can be implemented using any DL reasoner.

We then propose an algorithm to repair the *inconsistent mappings* in networked ontologies. Given two ontologies O_1 and O_2. Suppose $\mathcal{M}$ is a mapping between them which is inconsistent. We take the union O of O_1 and O_2 and assume that it is more reliable than mapping $\mathcal{M}$. Then we treat the problem of mapping repairing as the problem of DL-based ontology revision, i.e., problem of revising $\mathcal{M}$ by O. To revise $\mathcal{M}$, we apply our revision-based algorithm in possibilistic logic given in [Qi08] to find a diagnosis for the distributed system by treating $\mathcal{M}$ as a *possibilistic description logics knowledge base* (see [QPJ07] for details of possibilistic description logics).

Our evaluation on the algorithm for repairing mappings is done by considering the following evaluation measures: the first measure is the runtime of the algorithm and the second measure is the correctness or meaningfulness of the results of our approach.

Our experiment shows that our algorithm for finding MUPS is faster than the best algorithm for finding all MUPS of an ontology w.r.t. an unsatisfiable concept. We also compare our algorithm for mapping revision with the one given by Meilicke et.al. in [MST07] to show the efficiency and effectiveness of our algorithm.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The NeOn Big Picture

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

This report is part of the work performed in WP 1 on “Dynamics of Networked Ontologies”. The goal of this work package is to develop an integrated approach for the evolution process of networked ontologies and related metadata. As shown in Figure 1.1, WP1 belongs to the central part of the research and development WPs in NeOn. The tasks of WP1 are heavily inter-related with other work packages. For the individual phases of the process we will develop new methods that consider the complex relationships in a network of ontologies. These include dependencies, mappings, different versions and also take possible inconsistencies into account.

Specific goals in this workpackage include support for:

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

1.2 Motivation and Goals of this Deliverable

The problem of inconsistency (or incoherence) handling in ontologies has attracted a lot of attention. Inconsistency can occur due to several reasons, such as modeling errors, migration or merging ontologies, and ontology evolution. Many approaches were proposed to handle inconsistency in ontologies based on existing

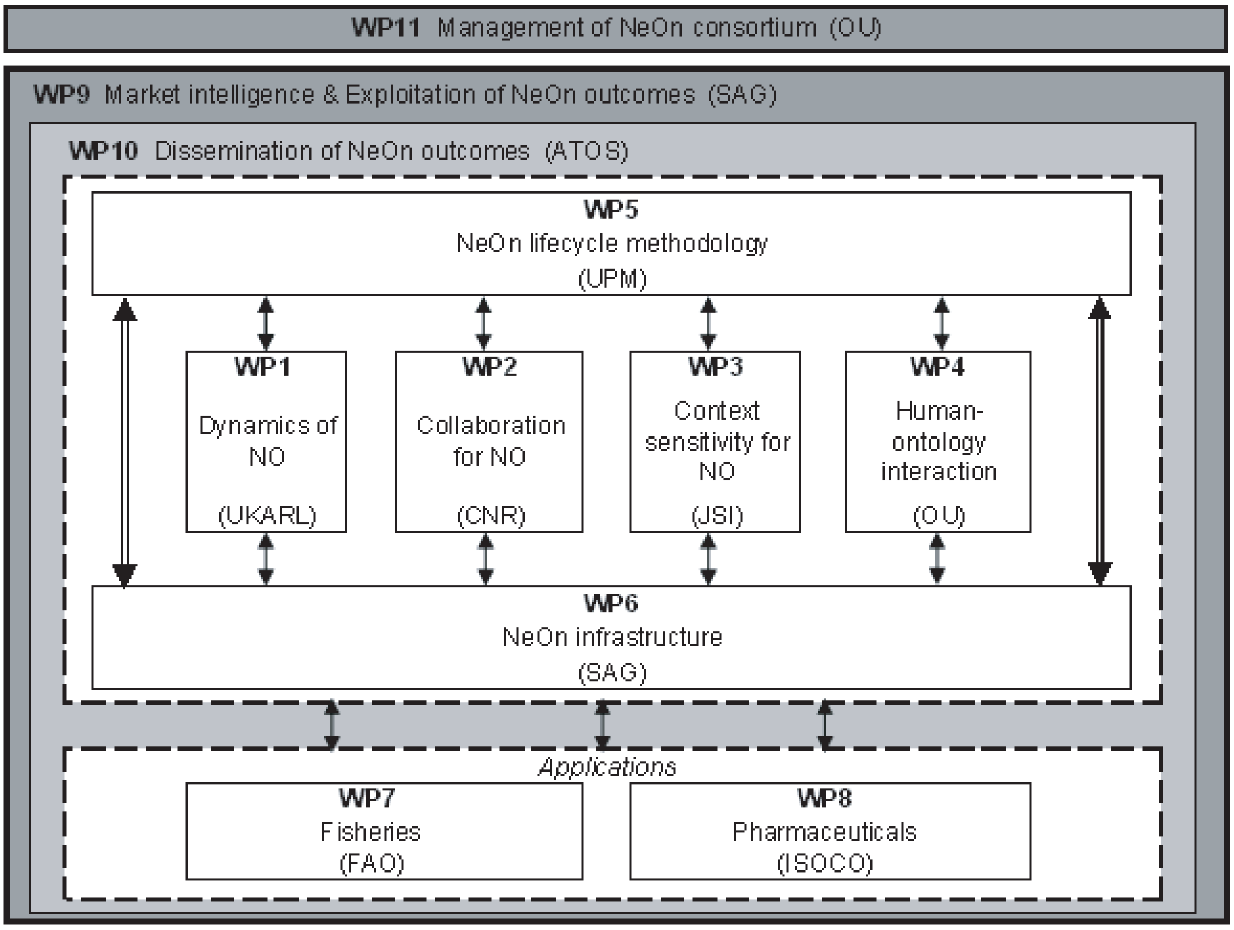
WP11 Management of NeOn consortium (OU)
WP9 Market intelligence & Exploitation of NeOn outcomes (SAG)
WP10 Dissemination of NeOn outcomes (ATOS)
WP5 NeOn lifecycle methodology (UPM)
WP1 Dynamics of NO (UKARL)
WP2 Collaboration for NO (CNR)
WP3 Context sensitivity for NO (JSI)
WP4 Human-ontology interaction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

Figure 1.1: Relationships between different workpackages in NeOn

techniques for inconsistency management in traditional logics, such as propositional logic and nonmonotonic logics [PSK05, HvHtT05]. In NeOn deliverable D1.2.2, we proposed a general approach for resolving inconsistency and incoherence in a single ontology and provided instantiation of the general approach. We also reported evaluation results of our instantiated approach. When resolving incoherence, we adapt the CS-Tree algorithm in [dlBSW03] for finding minimal unsatisfiable subsets to calculate all the *minimal unsatisfiability-preserving sub-ontologies (MUPS)*.

This deliverable continues the work given in D1.2.2. We mainly tackle the following two problems:

Firstly, several methods have been proposed to find all MUPS and then *minimal incoherence-preserving sub-TBox* (MIPS) (see [SC03, KPHS07, BPS07]). Although practical techniques to find all possible MUPS exist, efficiency is still a problem (see [SHCvH07]). As shown in [BPS07], in the worst case the number of MUPS for an unsatisfiable concept is exponential in the size of the ontology. Therefore, several approaches for resolving incoherence were given which do not calculate all the MUPS [Sch05, MST07].

To solve the above problem, we propose a novel algorithm for finding MUPS of an ontology w.r.t. an unsatisfiable concept by using a relevance-based selection function, called relevance-based algorithm. Our algorithm is based on a black-box approach, and thus it can be implemented using any DL reasoner. We first define a relevance-based ordering on the MUPS, which allows us to associate a relevance degree with each of the MUPS to facilitate the comparison among them. Specifically, we use a syntactic selection function based on concept relevance, whose intuition is to select axioms that are connected to the unsatisfiable concept to some extent. We then present an algorithm to find a set of MUPS for an unsatisfiable concept. The algorithm incrementally selects subsets of the ontology using the selection function and finds a set of MUPS from these sub-ontologies for the concept. When computing MUPS from a sub-ontology, our algorithm allows for different strategies: It either computes a set of MUPS that satisfies some condition(s) or computes all MUPS.

Secondly, theoretically, our approach given in D1.2.2 can be used to resolve inconsistencies in networked ontologies by considering the problem of mapping repair as the problem of ontology revision. However, the algorithm for mapping repair given in D1.2.2 requires to compute all the MUPS of an ontology w.r.t. an unsatisfiable concept. Therefore, it is not very efficient. In this deliverable, we propose a new algorithm for mapping revision based on the relevance-based algorithm for finding MUPS of an ontology w.r.t. an unsatisfiable concept.

The algorithms are implemented and evaluated by using the KAON2 reasoner[1] for the reasoning tasks. We also evaluate the efficiency and effectiveness of our algorithms using some real ontologies as data sets. By comparing our relevance-based algorithm and the algorithm given in [KPHS07], we show the advantage of introducing the selection function to find MUPS incrementally. We also show that our algorithm for mapping revision is better than the one given by Meilicke et.al. in [MST07] by comparing their efficiency and effectiveness.

The work presented in this deliverable is closely related to several activities in WP3. The confidence values attached to mappings can be viewed as a kind of provenance information, where provenance is a form of context that is typically available for automatically generated ontologies (see D3.1.3). Therefore, in our deliverable, we provide a practical usage of provenance to deal with inconsistency in networked ontologies. 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. Our approach for mapping revision provides a way to improve the quality of the automated generated mapping by the Alignment API.

1.3 Overview of the Deliverable

This deliverable is structured as follows. We first provide some background knowledge in Chapter 2. After that, we present our relevance-based algorithm in Chapter 3 and revision-based algorithm in Chapter 4. Experimental Results are reported in Chapter 5. Chapter 6 summarizes the conclusions and discusses future work.

[1] http://kaon2.semanticweb.org/

Chapter 2

Preliminaries

2.1 Debugging in Description Logics

We presume that the reader is familiar with Description Logics (DLs) and refer to the DL handbook [BCM$^+$03] for more details. A DL-based ontology (or knowledge base) $O = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a set $\mathcal{T}$ of concept axioms (TBox), a set $\mathcal{R}$ of role axioms (RBox), and a set $\mathcal{A}$ of assertional axioms (ABox). Concept axioms (or terminology axioms) have the form $C \sqsubseteq D$ where C and D are (possibly complex) concept descriptions, and role axioms are expressions of the form $R \sqsubseteq S$, where R and S are (possibly complex) role descriptions. The ABox contains *concept assertions* of the form $C(a)$ where C is a concept and a is an individual name, and *role assertions* of the form $R(a, b)$, where R is a role and a and b are individual names.

An interpretation $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty domain set $\triangle^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which maps from concepts and roles to subsets of the domain and binary relations on the domain, respectively. Given an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ satisfies a concept axiom $C \sqsubseteq D$ (resp., a role inclusion axiom $R \sqsubseteq S$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp., $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$). Furthermore, $\mathcal{I}$ satisfies a concept assertion $C(a)$ (resp., a role assertion $R(a, b)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is called a *model* of an ontology O, iff it satisfies each axiom in O. A concept C in an ontology O is unsatisfiable if for each model $\mathcal{I}$ of O, $C^{\mathcal{I}} = \emptyset$. An ontology O is incoherent if there exists an unsatisfiable concept in O.

Example 1 *Consider the following ontology (taken from the Proton ontology, c.f. experiments in Section 5)*
$O = \{$ *Manager* $\sqsubseteq$ *Employee, Employee* $\sqsubseteq$ *JobPosition, Leader* $\sqsubseteq$ *JobPosition, JobPosition* $\sqsubseteq$ *Situation, Situation* $\sqsubseteq$ *Happening, Leader* $\sqsubseteq \neg$ *Patent, Happening* $\sqsubseteq \neg$ *Manager, JobPostion* $\sqsubseteq \neg$ *Employee, JobPosition(lectureship)* $\}$.
There are two unsatisfiable concepts in this ontology: $Employee$ and $Manager$. Therefore, O is an incoherent ontology.

We generalize the notion of MUPS given in [SC03].

Definition 1 [1] *Let C be an unsatisfiable concept in an ontology O. A set $\mathcal{M} \subseteq O$ is a* minimal unsatisfiability-preserving sub-ontology (MUPS) *of O w.r.t. C if C is unsatisfiable in $\mathcal{M}$, and C is satisfiable in every sub-ontology $O' \subset \mathcal{M}$. The set of all MUPS of O w.r.t. C is denoted as $mups(C, O)$*

A MUPS of O w.r.t. C is a minimal sub-ontology of O in which C is unsatisfiable. It can be viewed as a justification of concept unsatisfiability [KPHS07]. Therefore, we sometimes call a MUPS as a justification.

Example 2 *In Example 1, concept $Manager$ has the following two MUPS:*
(1) $\mathcal{M}_1 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPostion $\sqsubseteq \neg$ Employee $\}$
(2) $\mathcal{M}_2 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq$ Situation, Situation $\sqsubseteq$ Happening, Happening $\sqsubseteq \neg$ Manager $\}$.

[1] MUPS is defined for a TBox in [SC03]. Its extension to an ontology is straightforward.

2.2 Selection Functions

We introduce the notion of selection function in a single ontology given in [HvHtT05], which will be used in our algorithm to extract a subset of an ontology relevant to a subsumption to some degree.

Definition 2 *(Selection Function) Let $\mathbf{L}$ be an ontology language (denoting sets of axioms), a selection function for $\mathbf{L}$ is a mapping $s_{\mathbf{L}}: \mathcal{P}(\mathbf{L}) \times \mathbf{L} \times \mathbb{N} \to \mathcal{P}(\mathbf{L})$ such that $s_{\mathbf{L}}(O, \phi, k) \subseteq O$, where $\mathcal{P}(\mathbf{L})$ is the power set of $\mathbf{L}$.*

That is, a selection function selects a subset of an ontology w.r.t. an axiom at step k. A syntactic relevance-based selection function is given as follows.

Let ϕ be an axiom in a DL-based ontology. We use $I(\phi)$, $C(\phi)$ and $R(\phi)$ to denote the sets of individual names, concept names, and role names appearing in ϕ respectively.

A specific selection function based on *syntactic relevance* is employed in our algorithm. We begin with defining *direct relevance* between two axioms.

Definition 3 *Two axioms ϕ and ψ are* directly relevant *iff there is a common name which appears both in ϕ and ψ, i.e., $I(\phi) \cap I(\psi) \neq \emptyset$ or $C(\phi) \cap C(\psi) \neq \emptyset$ or $R(\phi) \cap R(\psi) \neq \emptyset$.*

Based on the notion of direct relevance, we can define the notion of relevance between an axiom and an ontology.

Definition 4 *An axiom ϕ is relevant to an ontology O iff there exists an axiom ψ in O such that ϕ and ψ are directly relevant.*

We introduce the relevance-based selection function which can be used to find all the axioms in an ontology that are relevant to an axiom to some degree.

Definition 5 *Let O be an ontology, ϕ be an axiom and k be an integer. The* relevance-based selection function*, written s_{rel}, is defined inductively as follows:*

$s_{rel}(O, \phi, 0) = \emptyset$

$s_{rel}(O, \phi, 1) = \{\psi \in O : \phi \textit{ and } \psi \textit{ are directly relevant}\}$

$s_{rel}(O, \phi, k) = \{\psi \in O : \psi$ *is directly relevant to* $s_{rel}(O, \phi, k-1)\}$*, where* $k > 1$.

We call $s_{rel}(O, \phi, k)$ the k-relevant subset of O w.r.t. ϕ. For convenience, we define $s_k(O, \phi) = s_{rel}(O, \phi, k) \setminus s_{rel}(O, \phi, k-1)$ for $k \geq 1$.

To apply the relevance-based selection function to a concept, we need to construct a new axiom which states that this concept is sub-concept of a fresh concept which does not appear in the ontology. For notational simplicity, we use $s_{rel}(O, C, k)$ to denote $s_{rel}(O, C \sqsubseteq D, k)$ where D is a fresh concept. Similarly, $s_k(O, C)$ indicates $s_k(O, C \sqsubseteq D)$.

Example 3 *Consider the ontology given in Example 1.*

$O = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, Leader $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq$ Situation, Situation $\sqsubseteq$ Happening, Leader $\sqsubseteq \neg$ Patent, Happening $\sqsubseteq \neg$ Manager, JobPosition $\sqsubseteq \neg$ Employee, JobPosition(lectureship) $\}$.

We have

$s_1(O, Manager) = \{$ *Manager* $\sqsubseteq$ *Employee, Happening* $\sqsubseteq \neg$ *Manager* $\}$

$s_2(O, Manager) = \{$ *Employee* $\sqsubseteq$ *JobPosition, JobPosition* $\sqsubseteq \neg$ *Employee, Situation* $\sqsubseteq$ *Happening* $\}$

$s_3(O, Manager) = \{$ *Leader* $\sqsubseteq$ *JobPosition, JobPosition* $\sqsubseteq$ *Situation, JobPosition(lectureship)* $\}$

$s_4(O, Manager) = \{$ *Leader* $\sqsubseteq \neg$ *Patent* $\}$

2.3 Hitting Set Tree Algorithm

We briefly introduce some notions Reiter's Hitting Set Tree algorithm given in [Rei87] which will be used in our algorithm. We follow the reformulated notions in Reiter's theory given in [KPHS07]. Given a *universal set* U, and a set $\mathcal{C} = \{S_1, ..., S_n\}$ of subsets of U which are *conflict sets*, i.e. subsets of the system components responsible for the error. In the case of finding justifications, the universal set corresponds to ontology and a conflict set corresponds to a MUPS [KPHS07]. A *hitting set* T for $\mathcal{C}$ is a subset of U such that $S_i \cap T \neq \emptyset$ for all $1 \leq i \leq n$. A hitting set is used to repair conflicts. That is, if we remove elements in T from U, then we resolve all the conflict sets in $\mathcal{C}$. In practice, we want to remove as little information as possible to resolve conflicts. This can be captured by the notion of a *minimal hitting set*. A minimal hitting set T for $\mathcal{C}$ is a hitting set such that no $T' \subset T$ is a hitting set for $\mathcal{C}$. A minimal hitting set is a hitting set which is minimal w.r.t. a set-inclusion relation. Alternatively, we can define the minimality based on the cardinality of a set. A hitting set T is cardinality-minimal if there is no other hitting set T' such that $|T'| < |T|$.

Reiter's algorithm is used to calculate minimal hitting sets for a collection $\mathcal{C} = \{S_1, ..., S_n\}$ of sets by constructing a labeled tree, called a Hitting Set Tree (HST). Given a collection $\mathcal{C}$ of sets, a HST T is the smallest edge-labeled and node-labeled tree, such that the root is labeled by $\checkmark$ if $\mathcal{C}$ is empty. Otherwise it is labeled with any set in $\mathcal{C}$. For each node n in T, let $H(n)$ be the set of edge labels on the path in T from the root to n. The label for n is any set $S \in \mathcal{C}$ such that $S \cap H(n) = \emptyset$, if such a set exists. If n is labeled by a set S, then for each $\sigma \in S$, n has a successor, n_σ joined to n by an edge labeled by σ. For any node labeled by $\checkmark$, $H(n)$, i.e. the labels of its path from the root, is a hitting set for $\mathcal{C}$. The HST algorithm usually results in several minimal hitting sets. We randomly choose one of them as output of the algorithm.

Figure 2.1 shows a HST T for the collection $\mathcal{C} = \{\{1,2,3,4,5,6\},\ \{3,4,5\},\ \{1,2,4,6\},\ \{1,2\},\ \{4,7\}\}$ of sets. T is created breadth first, starting with root node n_0 labeled with $\{1,2,3,4,5,6\}$. For diagnostic problems the sets in the collection are conflict sets which are created on demand. In our case, conflict sets for a terminological diagnosis problem can be calculated by a standard DL engine (by definition each incoherent subset of $\mathcal{T}$ is a conflict set).

2.4 Diagnosing and Repairing Mappings

In this section, we introduce some notions of diagnosing and repairing mappings, which have been given in [MST07].

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+07].

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 6 *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 6, 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+07].

Definition 7 *Given ontologies O_1 and O_2, let Q be a function that is given in Definition 6. $\mathcal{M}$ 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 $e' \in Q(O_2)$.*

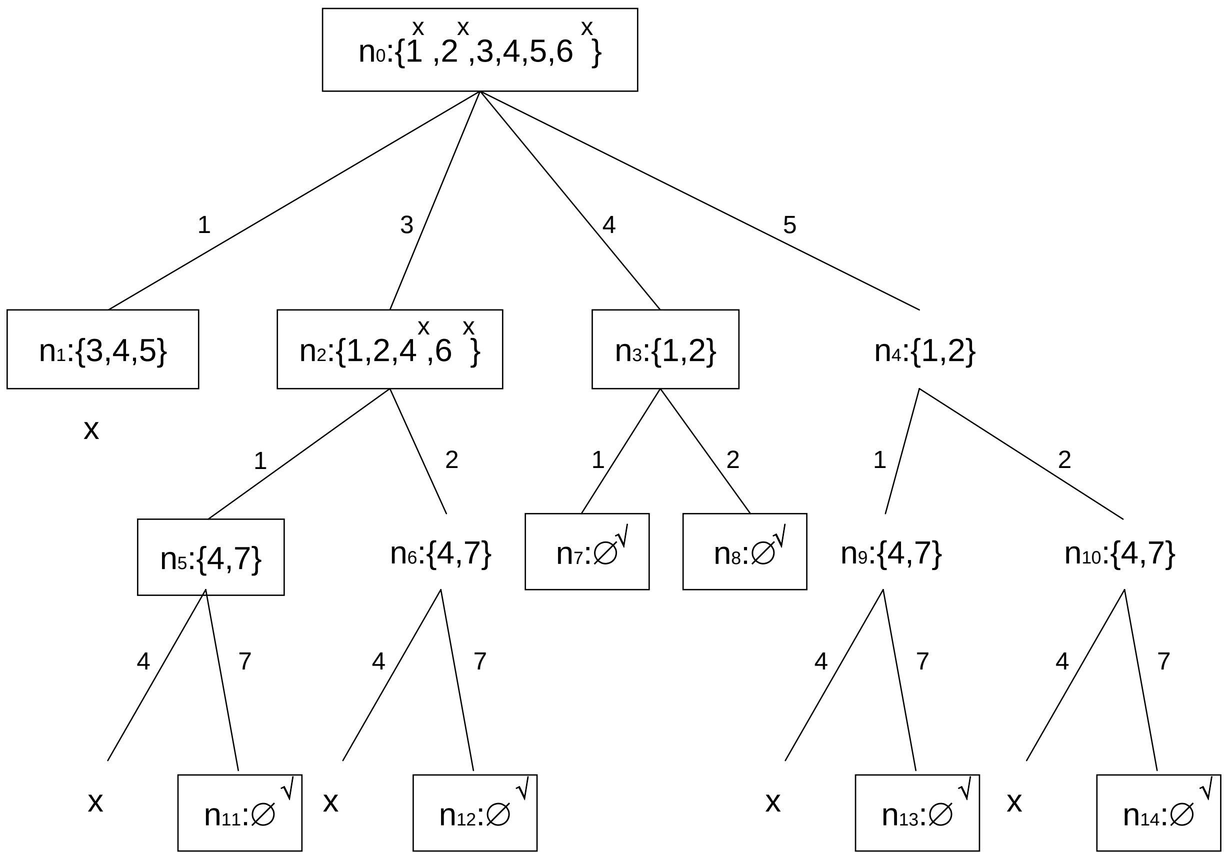

Figure 2.1: HST with small conflict sets

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

Given a mapping between two ontologies O_1 and O_2, we can define the notion of a distributed system[2] [ZE06].

Definition 8 *A distributed system is a triple $\mathcal{D} = \langle O_1, O_2, \mathcal{M} \rangle$, where O_1 and O_2 are ontologies and $\mathcal{M}$ is a mapping between them. We call O_1 the source ontology and O_2 the target ontology.*

Given a distributed system, there is no unique semantics for it. Three semantics for distributed systems have been proposed in [ZE06]. While theoretically interesting, none of their semantics has been used in practice. Another important semantics for a distributed system is defined by Distributed Description Logics (DDL). Algorithms for reasoning tasks in DDL, such as consistency checking, have been implemented as an extension DRAGO system [ST05]. Although the complexity of reasoning in a DDL is the same as that of reasoning in the local language of the DDL, the implementation DRAGO system lacks optimization so that the system runs slowly for real life ontologies. In our work, we follow the notion of consistency of mapping given in [MS07]. This semantics is the same as that of the mapping system described in [HM05], which has been discussed in NeOn deliverable D1.1.2 [HBP$^+$07].

Example 4 *Take the two ontologies CRS and EKAW in the domain of conference management systems as an example, which are real-life ontologies provided by OAEI conference [3]. For simplicity, we use a subset of*

[2] In [ZE06], they do not restrict to two ontologies. To simplify discussions, we consider only two ontologies in our work.
[3] http://oaei.ontologymatching.org/2007/

each ontology in our example. The source ontology O_1 with namespace $ns1$ and the target ontology O_2 with namespace $ns2$ contain the following axioms:

$$\begin{aligned}
&ns1 : Paper \sqsubseteq ns1 : Document, \quad ns1 : PC\_Member \sqsubseteq ns1 : Possible\_Review \\
&ns1 : Conference\_Paper \sqsubseteq ns1 : Paper, \quad ns1 : Workshop\_Paper \sqsubseteq ns1 : Paper, \\
&ns2 : article \sqsubseteq ns2 : document, \quad ns2 : program \sqsubseteq \neg ns2 : document,
\end{aligned} \tag{2.1}$$

The mapping $\mathcal{M}$ between O_1 and O_2 is obtained by the ontology matching system HMatch and is a subset of the mapping which is available online [4]*. The correspondences in $\mathcal{M}$ are listed as follows:*

$$\begin{aligned}
&m_1 : \langle ns2 : article, ns1 : Conference\_Paper, \sqsubseteq, 0.65\rangle, \\
&m_2 : \langle ns1 : Workshop\_Paper, ns2 : article, \sqsubseteq, 0.65\rangle, \\
&m_3 : \langle ns1 : Document, ns2 : program, \sqsubseteq, 0.8\rangle, \\
&m_4 : \langle ns2 : program, ns1 : Document, \sqsubseteq, 0.8\rangle, \\
&m_5 : \langle ns2 : document, ns1 : Document, \sqsubseteq, 0.93\rangle
\end{aligned} \tag{2.2}$$

Definition 9 *Let O_1 and O_2 be two ontologies and $\mathcal{M}$ be a mapping between them. The* union *$O_1 \cup_{\mathcal{M}} O_2$ of O_1 and O_2 connected by $\mathcal{M}$ is defined as $O_1 \cup_{\mathcal{M}} O_2 = O_1 \cup O_2 \cup \{t(x) : x \in \mathcal{M}\}$ with t being a translation function that converts correspondences into axiom in the following way:*

$$t(\langle C, C', r, \alpha\rangle) = CrC'$$

That is, we first translate all the correspondences in the mapping $\mathcal{M}$ into DL axioms, then the union of the two ontologies connected by the mapping is the set-union of the two ontologies and the translated axioms. Take a correspondence in Example 4 as an example. $t(\langle ns2 : article, ns1 : Conference\_Paper, \sqsubseteq, 0.4\rangle) = ns2 : article \sqsubseteq ns1 : Conference\_Paper$.

Definition 10 *Given two ontologies O_1 and O_2 and a mapping $\mathcal{M}$ between them. Then $\mathcal{M}$ is consistent iff there exists no concept C in O_i with $i \in \{1, 2\}$ such that C is satisfiable in O_i but unsatisfiable in $O_1 \cup_{\mathcal{M}} O_2$. Otherwise, $\mathcal{M}$ is inconsistent.*

An inconsistent mapping is a mapping such that there is a concept that is satisfiable in a local ontology but unsatisfiable in the union of the two ontologies connected by the mapping. It has been argued in [MS07] and [MST07] that an inconsistent mapping is a mapping that contains erroneous correspondences. In Example 4, the mapping $\mathcal{M}$ is inconsistent since there are three unsatisfiable concepts $ns1 : Workshop\_Paper$, $ns2 : article$ and $ns2 : document$ in $O_1 \cup_{\mathcal{M}} O_2$ which are satisfiable in both O_1 and O_2.

In [MST07], the notion of diagnosis for a classical knowledge base is applied to a distributed system. We adapt their definition as follows.

Definition 11 *Given a distributed system $\langle O_1, O_2, \mathcal{M}\rangle$, where O_1 and O_2 are ontologies and $\mathcal{M}$ is a mapping between them. A diagnosis for the distributed system is the minimal set $\mathcal{M}' \subseteq \mathcal{M}$ such that $\mathcal{M} \setminus \mathcal{M}'$ is consistent w.r.t. O_1 and O_2.*

Intuitively speaking, a diagnosis for a distributed system is a minimal set of correspondences such that the mapping in the distributed system becomes consistent after removing these correspondences. Following Example 4, both $\{ns1 : Document \sqsubseteq ns2 : program\}$ and $\{ns2 : article \sqsubseteq ns1 : Conference\_Paper,\ ns1 : Workshop\_Paper \sqsubseteq ns2 : article,\ ns2 : document \sqsubseteq ns1 : Document\}$ are diagnoses for the distributed system $\langle O_1, O_2, \mathcal{M}\rangle$.

We adapt the notion of a minimal conflict set of a distributed system given in [MST07] as follows.

[4]http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html

Definition 12 *Given a distributed system $\langle O_1, O_2, \mathcal{M}\rangle$, where O_1 and O_2 are ontologies and $\mathcal{M}$ is a mapping between them. Suppose $\mathcal{M}$ is an inconsistent mapping. A subset $\mathcal{C}$ of $\mathcal{M}$ is a* conflict set *for a concept A in O_i ($i = 1, 2$) iff A is satisfiable in O_i but unsatisfiable in $O_1 \cup_{\mathcal{C}} O_2$. $\mathcal{C}$ is a* minimal conflict set *for A in O_i iff $\mathcal{C}$ is a conflict set for A and there exists no $\mathcal{C}' \subset \mathcal{C}$ which is also a conflict set for A in O_i.*

A minimal conflict set for a concept in one of the ontologies is a minimal subset of the mapping that, together with the local ontologies, is responsible for the unsatisfiability of the concept in the distributed system. It is similar to the notion of a MUPS of a single ontology w.r.t. an unsatisfiable concept. For example, $\mathcal{C} = \{ns1 : Document \sqsubseteq ns2 : program,\ ns1 : Workshop\_Paper \sqsubseteq ns2 : article\}$ is a minimal conflict set for concept $Workshop\_Paper$ in O_1.

Chapter 3

A Relevance-based Algorithm for Finding MUPS

In this chapter, we first introduce a relevance-based ordering on MUPS and then present an algorithm that computes a set of MUPS attached with relevance degrees.

3.1 Relevance-based Ordering on MUPS

We define an ordering on MUPS using the relevance-based selection function.

Definition 13 *Let O be an ontology and C be an unsatisfiable concept in O. s_{rel} is the relevance-based selection function given by Definition 5. A relevance-based ordering on the set of all the MUPS for C, written $\preceq_{rel,C}$, is defined as follows: for any two MUPS J_1 and J_2 for C,*

$$J_1 \preceq_{rel,C} J_2 \quad iff \quad d_{rel,C}(J_1) \geq d_{rel,C}(J_2)$$

where, $d_{rel,C}(J_i)$ = $max\{k : J_i \cap s_k(O,C) \neq \emptyset\}$ is used to measure the relevance degree of J_i w.r.t. C.

That is, MUPS J_1 is less relevant to C than J_2 if and only if the element in J_1 which is furthest from C is less relevant to that in J_2 which is furthest from C.

Example 5 *(Example 3 Continued) Concept $Manager$ has the following two MUPS:*
(1) $J_1 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq \neg$ Employee $\}$
(2) $J_2 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq$ Situation, Situation $\sqsubseteq$ Happening, Happening $\sqsubseteq \neg$ Manager $\}$.
By Example 3, we have $d_{rel,Manager}(J_1) = 2$ and $d_{rel,Manager}(J_2) = 3$. Therefore, $J_2 \preceq_{rel,Manager} J_1$. It is clear that J_1 is much easier to understand than J_2.

Let us consider an important property of the relevance-based ordering:

Proposition 1 *Let O be an ontology and C be an unsatisfiable concept in O. Given a MUPS J for C, suppose $k = d_{rel,C}(J)$, then $J \cap s_j(O,C) \neq \emptyset$ for any $0 < j \leq k$.*

Proof 1 *It is clear that $J \cap s_k(O,C) \neq \emptyset$. Suppose that $J \cap s_j(O,C) = \emptyset$ for some $j < k$. Then for any axiom ϕ in $s_{rel}(O,C,j-1) \cap J$ and any axiom ψ in $J \setminus s_{rel}(O,C,j-1)$, ϕ and ψ are not directly relevant because ϕ can be only directly relevant to axioms in $s_{rel}(O,C,j-1)$ or axioms in $s_j(O,C)$. We show that J cannot be a MUPS. Let us construct an ontology O' which contains exact those elements of J. According to [KPSH05], any MUPS for an unsatisfiable concept must be generated by a black-box algorithm for finding* *a single MUPS. Therefore J must be generated by the black-box algorithm. In the black-box algorithm, relevance-based selection function s_{rel} is applied to select axioms from O'. It is clear that the algorithm will select axioms from $s_1(O,C) \cap O'$, $s_2(O,C) \cap O'$, etc. However, it will stop selecting axiom when it reaches $s_{j-1}(O,C) \cap O'$ because there is no element in $s_j(O,C) \cap O'$. So the MUPS found by the algorithm will not include any axiom in $J \setminus s_{rel}(O,C,j-1)$. Contradiction.*

The Proposition 1 states that if the relevance degree of a MUPS for a concept is k, then it contains at least one axiom in the ontology that is relevant to the concept with degree j for any $0 < j < k$.

Corollary 1 *Let O be an ontology and C be an unsatisfiable concept in O. Given two MUPS J_1 and J_2 for C, suppose $J_1 \preceq_{rel,C} J_2$, if $J_2 \cap s_k(O,C) \neq \emptyset$ then $J_1 \cap s_k(O,C) \neq \emptyset$ for all k, but not vice versa.*

Corollary 1 follows from Proposition 1. According to Corollary 1, suppose MUPS J_1 is less relevant to C than J_2, if J_2 is k-relevant to C, then J_1 must be k-relevant to C as well, where a MUPS is k-relevant to C if and only if it has non-empty intersection with $s_k(O,C)$. Proposition 1 and Corollary 1 together tell us that if we want to find MUPS for a concept in an ontology that are more relevant to the concept, we should select those axioms in the ontology that are more relevant to the concept.

3.2 Relevance-based Algorithm for Finding MUPS

Our algorithm receives an ontology O, an unsatisfiable concept C in O and a strategy for computing MUPS as inputs, and outputs a set of weighted MUPS $\overrightarrow{\mathcal{J}}$ and a set of hitting sets HS for $mups(C,O)$, which we call global hitting sets. In our algorithm, we consider two strategies when expanding the hitting set tree by invoking Algorithm 2: (1) All_Just_Relevance is to compute all the MUPS when we expand the HST by using Algorithm 2, and (2) CM_Just_Relevance computes those MUPS that are in nodes of the cardinality-minimal hitting sets in the local HST[1] expanded by Algorithm 2.

First of all, we find the first k such that C is unsatisfiable in the k-relevant subset O' of O, i.e., the “if" condition in line 19 is satisfied. We then call Algorithm 2 to find a set of MUPS for C in O' and a set of local hitting sets for $mups(C,O')$. After that, we associate a relevance degree to each found MUPS in line 21.

We then add to O' axioms in O that are directly relevant to O'. Since HS_{local} is not empty, the “if" condition in line 7 is always satisfied. In the first “for" loop, we get all the hitting sets that are global hitting sets, i.e., C becomes satisfiable if we remove axioms in such a hitting set from O (lines 8-10). We exclude from HS_{local} those hitting sets that have been selected (line 11). Now if we choose the strategy CM_Just_Relevance and there exists a global hitting set in HS, then the algorithm terminates and returns the found weighted MUPS and global hitting sets. Alternatively, if HS_{local} is empty, then there is no hitting set tree to expand anymore. So the algorithm terminates and returns the found weighted MUPS and hitting sets (lines 12-13). If none of the conditions in line 12 is satisfied, we expand the hitting set tree in lines 15-18. For each local hitting set P that has been found before, we call Algorithm 2 to find a set of MUPS for C in $O' \setminus P$ and a set of local hitting sets for $mups(C, O' \setminus P)$ (line 16). After that, we update the set of weighted MUPS and the set of local hitting sets (lines 17-18).

In Algorithm 2, we first find a single MUPS (line 3) and add it to $\mathcal{J}$ which contains all the MUPS generated by the algorithm (line 4). We then create all the possible branches from the node corresponding to the new found MUPS (lines 5-6). In the “while" loop, we first find all the branches that do not need to expand and all the branches to be expanded (steps 9-14). Our algorithm terminates and returns the found MUPS and hitting sets if any of the following condition holds: (1) The strategy CM_Just_Relevance is chosen and we have found a local hitting set, (2) the input concept C is satisfiable in the input ontology (i.e. $HS_1 = \emptyset$ in the first iteration of “while" loop), and (3) there is no branch to expand (i.e. $HS_2 = \emptyset$). If none of the conditions is satisfied, then we expand the branches stored in HS_2. For each $P \in HS_2$, we first find a single MUPS for

[1]We call the HST constructed using Algorithm 2 as a local HST, and the hitting sets found in this algorithm as local hitting sets.

Algorithm 1: REL_JUSTS(C,O,$compute\_justs\_strategy$)

```
Data: An ontology $O$ and an unsatisfiable concept $C$ of $O$, and the strategy $compute\_justs\_strategy$ to
      compute MUPS.
Result: A set of weighted MUPS $\overrightarrow{\mathcal{J}}$ and a set of global hitting sets $HS$
begin
    Globals : $strategy \leftarrow compute\_justs\_strategy$; $\overrightarrow{\mathcal{J}} \leftarrow \emptyset$
    $O' \leftarrow HS \leftarrow HS_{local} \leftarrow \emptyset$; $k \leftarrow 1$
    $\mathcal{S}_{rel} \leftarrow s_k(O, C)$
    while $\mathcal{S}_{rel} \neq \emptyset$ do
        $O' \leftarrow O' \cup \mathcal{S}_{rel}$
        if $HS_{local} \neq \emptyset$ then
            for $P \in HS_{local}$ do                        /* Get global hitting sets */
                if $C$ is satisfiable in $O \setminus P$ then
                    $HS \leftarrow HS \cup \{P\}$
            $HS_{local} \leftarrow \{P | P \in HS_{local}$ and $P \notin HS\}$
            if ($strategy$ ≠All_Just_Relevance and $HS \neq \emptyset$) or ($HS_{local} = \emptyset$) then
                return $(\overrightarrow{\mathcal{J}}, HS)$                  /* Early termination */
            $HS_{temp} \leftarrow HS_{local}$
            for $P \in HS_{temp}$ do                         /* Expand hitting set tree */
                $(\mathcal{J}, HS'_{local}) \leftarrow$ COMPUTE_JUSTS$(C, O' \setminus P)$
                $\overrightarrow{\mathcal{J}} \leftarrow \overrightarrow{\mathcal{J}} \cup \{(J,k) | J \in$ newly found MUPS$\}$
                $HS_{local} \leftarrow HS_{local} \cup \{P \cup P' | P' \in HS'_{local}\} \setminus \{P\}$
        else if $C$ is unsatisfiable in $O'$ then
            $(\mathcal{J}, HS_{local}) \leftarrow$ COMPUTE_JUSTS$(C, O')$
            $\overrightarrow{\mathcal{J}} \leftarrow \{(J,k) | J \in \mathcal{J}\}$     /* Associate relevance degree */
        $k \leftarrow k + 1$
        $\mathcal{S}_{rel} \leftarrow s_k(O, C)$
    return $(\overrightarrow{\mathcal{J}}, HS)$
end
```

C in $O \setminus P$ and add it to $\mathcal{J}$ (steps 19 and 20). We then create all the possible new branches (lines 21-22) and go to another iteration of “while" loop.

To compute a single MUPS, we can take the methods given in [KPHS07]. For example, SINGLE_JUST(C, O) in Algorithm 2 can be the black-box algorithm in [KPHS07]. It first expands a freshly generated ontology O' which is a superset of a MUPS using a relevance-based selection function. Then O' is pruned to find the final MUPS, where a window-based pruning strategy is used to minimize the number of satisfiability-check calls to the reasoner.

We show the correctness of Algorithm 1 when the strategy All_Just_Relevance is chosen, i.e., our algorithm finds all the MUPS for $mups(C, O)$ when it terminates.

Theorem 1 *Let the strategy to compute MUPS in Algorithm 1 be to compute all MUPS ($compute\_justs\_strategy = all\_Justification$). Suppose $\overrightarrow{\mathcal{J}}$ is the set of weighted MUPS returned by REL_JUSTS($C, O,$ $compute\_justs\_strategy$), then $mups(C, O) = \{J | \exists k, (J, k) \in \overrightarrow{\mathcal{J}}\}$. For each $(J, k) \in \overrightarrow{\mathcal{J}}$, we have $k = d_{rel,C}(\mathcal{J})$.*

Proof 2 *(sketch) It is easy to check that Algorithm 1 will terminate and return $\overrightarrow{\mathcal{J}} = \emptyset$ when C is satisfiable in O. So we assume that C is unsatisfiable in O. Since it has been shown that HST algorithm can be used to*

Algorithm 2: COMPUTE_JUSTS(C,O)

```
Data: An ontology $O$ and an unsatisfiable concept $C$ of $O$
Result: A set of MUPS $\mathcal{J}$ for $C$ in $O$ and a set of hitting sets $HS$
1  begin
2      $HS \leftarrow HS_1 \leftarrow \emptyset$
3      $J \leftarrow$ SINGLE_JUST$(C, O)$
4      $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
5      for $a \in J$ do                                  /* Create all possible branches. */
6          $HS_1 \leftarrow HS_1 \cup \{\{a\}\}$
7      while true do
8          $HS_2 \leftarrow \emptyset$
9          for $(P \in HS_1)$ do
10             if $C$ is satisfiable in $O \setminus P$ then            /* Local hitting sets */
11                 $HS \leftarrow HS \cup \{P\}$
12             else                                 /* Branches need to be expanded */
13                 $HS_2 \leftarrow HS_2 \cup \{P\}$
14
15         if ($strategy$ ≠All_Just_Relevance and $HS \neq \emptyset$) or ($HS_1 = \emptyset$) or ($HS_2 = \emptyset$) then
16             return $(\mathcal{J}, HS)$
17         $HS_1 \leftarrow \emptyset$
18         for $P \in HS_2$ do
19             $J \leftarrow$ SINGLE_JUST$(C, O \setminus P)$
20             $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
21             for $a \in J$ do
22                 $HS_1 \leftarrow HS_1 \cup \{P \cup \{a\}\}$
23 end
```

find all the MUPS in [KPHS07], we only need to show that Algorithm 1 expand all the branches of the hitting set tree for $mups(C, O)$. This can be seen from the following facts: (1) for each local hitting set in HS_{local}, if it is a global hitting set, then it is stored in the set HS (lines 8-10); (2) Early termination will not happen unless there is no local hitting set to expand, i.e., we have found all the global hitting sets; (3) in Algorithm 2, since we have chosen the strategy All_Just_Relevance, the algorithm terminates only if there is no more branch that can be expanded, i.e., either $HS_1 = \emptyset$ or $HS_2 = \emptyset$. Since we add all the possible branches to HS_1 (lines 5-7 and lines 22-24), all the branches have been explored when the algorithm terminates. It is clear that $k = d_{rel,C}(\mathcal{J})$ according to steps 17 and 21 of Algorithm 1.

We show the correctness of Algorithm 2 when the strategy CM_Just_Relevance is chosen, i.e., it computes those MUPS that are nodes of the cardinality-minimal hitting sets in the local hitting set tree expanded by Algorithm 2.

Theorem 2 *Suppose strategy CM_Just_Relevance is an input of Algorithm 1. Suppose $\mathcal{J}$ and HS are returned by COMPUTE_JUSTS(C,O), then for each local hitting set $J \in \mathcal{J}$, there is no other local hitting set J' in $mups(C, O)$ such that $|J'| < |J|$.*

Proof 3 *Since we choose strategy CM_Just_Relevance, there are three cases where the algorithm terminates (lines 15-16). Case 1: $HS \neq \emptyset$, i.e., there is a hitting set in $mups(C, O)$. In this case, all the hitting sets in HS has the same cardinality l. It is impossible to find a hitting set whose cardinality is less than l. Otherwise, since we expand the hitting set in the breadth-first manner, this hitting set must have been put*

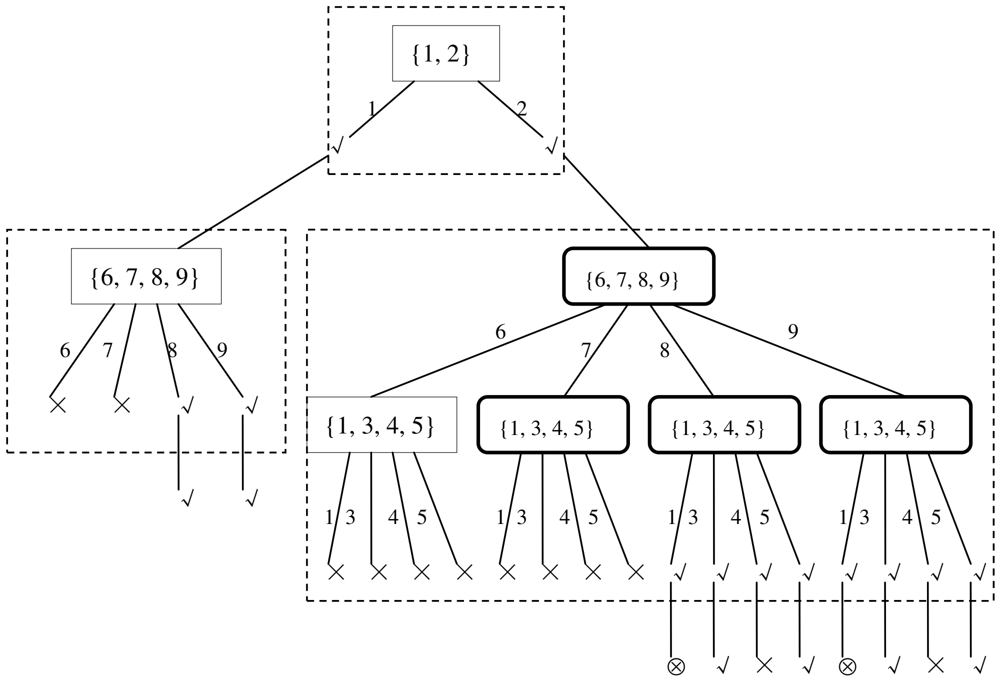

Figure 3.1: **Finding MUPS using relevance-based algorithm**

in HS before and the algorithm should have terminated. This is a contradiction. Case 2: $HS_1 = \emptyset$. In this case, we must have that there is no MUPS for C in O, so $mups(C, O) = \emptyset = \mathcal{J}$. Case 3: $HS_2 = \emptyset$. In this case, we have expanded all the branches and we must have found a hitting set, i.e., condition $HS \neq \emptyset$ is satisfied. So the conclusion holds.

We go through our algorithm to compute some MUPS by considering the following example.

Example 6 *Given an ontology including the following axioms:*

1: $U \sqsubseteq A$,	*2:* $U \sqsubseteq \neg A$,	*3:* $U \sqsubseteq C$,	*4:* $C \sqsubseteq \neg B$,
5: $A \sqsubseteq B$,	*6:* $U \sqsubseteq G$,	*7:* $G \sqsubseteq E$,	*8:* $U \sqsubseteq F$,
9: $F \sqsubseteq \neg E$,	*10:* $U \sqsubseteq D$,	*11:* $D \sqsubseteq E$,	*12:* $C \sqsubseteq K$,
13: $K \sqsubseteq \neg H$,	*14:* $B \sqsubseteq H$		

We denote each axiom by a natural number (1..14) for simplicity. In O there is only one unsatisfiable concept U. For the concept U, we obtain the following subsets of O using selection function: $s_1(O, U) = \{1, 2, 3, 6, 8, 10\}$, $s_2(O, U) = \{4, 5, 7, 9, 11, 12\}$ and $s_3(O, U) = \{13, 14\}$. All the MUPS for U are listed as follows:

$\mathcal{J} = \{\{1, 2\}, \{1, 3, 4, 5\}, \{6, 7, 8, 9\}, \{8, 9, 10, 11\}, \{1, 3, 5, 12, 13, 14\}\}$.

We illustrate our algorithm where CM_Just_Relevance is chosen in Figure 3.1, where each distinct node in a rectangular box represents a MUPS, and each sub-tree outlined in a rectangular box shows the process to compute MUPS for a sub-ontology by invoking Algorithm 2.

***k = 1** : $O' = s_{rel}(O, U, 1)$. A MUPS $J = \{1, 2\}$ is computed by Algorithm 2 and set as the root node of the hitting set tree (HST) (see the top rectangular box in Figure 3.1). Then two possible branches $\{1\}$ and $\{2\}$ can be generated from J (see lines 5-7 of Algorithm 2). Since U turns satisfiable in $O' \setminus \{1\}$ and $O' \setminus \{2\}$ (see lines 10-14), we know the two branches are local hitting sets and then go back to Algorithm 1 (see lines 15-16). We then associate relevance degree 1 to each returned MUPS in $HS = \{\{1\}, \{2\}\}$ (line 22 of Algorithm 1).*

***k = 2** : $O' = s_{rel}(O, U, 2)$. Since no local hitting sets are global (see lines 7-10 of algorithm 1), we need to expand each branch stored in HS.*

Take branch $\{2\}$ as an example (suppose we have expanded branch $\{1\}$ in the left rectangular box below the root node). We call Algorithm 2 with inputs U and $O' \setminus \{2\}$. The MUPS $J_1 = \{6, 7, 8, 9\}$ which has been computed before is reused and is set as the root node of local HST for $mups(U, O' \setminus \{2\})$. Check each possible branch originated from J_1, none of them is a local hitting set and so we need to generate a new MUPS along each branch (see lines 18-21). For example, we generate $J_2 = \{1, 3, 4, 5\}$ along branch $\{6\}$ and construct new branches $\{6, 1\}$, $\{6, 3\}$, $\{6, 4\}$ and $\{6, 5\}$. Similarly other branches $\{7\}$, $\{8\}$ and $\{9\}$ in $O' \setminus \{2\}$ can be generated and expanded. We then go to next iteration to check each newly generated branch and denote those local hitting sets with '$\surd$' (otherwise, with '$\times$'). From the figure we can see that 8 local hitting sets are found in $O' \setminus \{2\}$. Then we go back to Algorithm 1 and associate relevance degree 2 to the newly computed MUPS $\{1, 3, 4, 5\}$. Also, we replace the prefix-branch $\{2\}$ with new paths by combining the newly found local hitting sets with $\{2\}$ in line 18 of Algorithm 1.

***k = 3** : $O' = s_{rel}(O, U, 3)$. In this iteration, we check all possible branches and find 8 global hitting sets which are marked with '$\surd$' outside the rectangular box. The condition in line 12 of Algorithm 1 is satisfied, so the algorithm terminates and outputs global hitting sets $HS = \{\{1, 8\}, \{1, 9\}, \{2, 8, 3\}, \{2, 8, 5\}, \{2, 9, 3\}, \{2, 9, 5\}\}$
and a set of weighted MUPS $\overrightarrow{\mathcal{J}} = \{(\{1, 2\}, 1), (\{6, 7, 8, 9\}, 2), (\{1, 3, 4, 5\}, 2)\}$.*

Since our relevance-based algorithm is based on modified HST algorithm [Rei87], similar to [KPHS07], we can apply the optimized techniques in Reiter's HST algorithm in our algorithm: We can apply *MUPS reuse*, which means if the current edge path in any branch of the HST does not overlap with a MUPS, then this MUPS can be used as a new node to this branch. This optimization can be applied in the following ways. First, some MUPS found in previous iterations may be reused. For example (see Figure 3.1), the MUPS $\{6, 7, 8, 9\}$ in oval border is reused among different sub-ontologies: one sub-ontology is $s_{rel}(O, U, 2) \setminus \{1\}$ and another one is $s_{rel}(O, U, 2) \setminus \{2\}$. Second, in one sub-ontology, it is possible to reuse some previously found MUPS along different branches. For example, in the sub-ontology $s_{rel}(O, U, 2) \setminus \{2\}$, MUPS $\{1, 3, 4, 5\}$ is reused three times since this MUPS has no interaction with each branch (i.e. $\{6\}$, $\{7\}$,$\{8\}$ or $\{9\}$).

Early path termination can be applied to our algorithm in different ways. First, if the current edge path in any branch of the HST is a superset of some hitting set, this path is a hitting set as well and it is not necessary to further expand this branch. For example, the path $\{2, 8, 1\}$ marked with $\otimes$ in Figure 3.1 is a superset of $\{1, 8\}$ which is a previously found hitting set. Second, the current edge can be terminated if it is a prefix-path of some found hitting set. For instance, path $\{a_1, a_2\}$ is a prefix-path of a hitting set $\{a_1, a_2, a_3\}$. Finally, if strategy CM_Just_Relevance is chosen in our algorithm, once we found some local hitting sets, the other branches correspond to non-local hitting sets can be terminated. See the figure again, the branches marked with '$\times$' in each rectangular box are not expanded anymore.

Chapter 4

A Revision-based Algorithm for Mapping Repair

In this chapter, we present a revision-based algorithm for repairing mappings. The idea is described as follows. Given two ontologies O_1 and O_2. Suppose $\mathcal{M}$ is a mapping between them which is inconsistent. We take the union O of O_1 and O_2 and assume that it is more reliable than the mapping $\mathcal{M}$. Then we treat the problem of mapping repair as the problem of DL-based ontology revision (this has been pointed out in [QHH$^+$08] and [MST08]), i.e., problem of revising $\mathcal{M}$ by O. To revise $\mathcal{M}$, we apply our revision-based algorithm given in [Qi08] to find a diagnosis for the distributed system by treating $\mathcal{M}$ as a possibilistic DL knowledge base [1][QPJ07]. More specifically, given a mapping $\mathcal{M}$, we can translate it into a possibilistic DL knowledge base $O_{\mathcal{M}}$ using the translation function defined in Definition 9: $O_{\mathcal{M}} = \{(t(\langle C, C', r, \alpha\rangle), \alpha) : \langle C, C', r, \alpha\rangle \in \mathcal{M}\}$.

We need to define the notion of an inconsistency degree of a distributed system, which will be used to define our algorithm.

Definition 14 *Given a distributed system $\mathcal{D} = \langle O_1, O_2, \mathcal{M}\rangle$, the β-cut (resp. strict β-cut) set of $\mathcal{D}$, denoted as $\mathcal{D}_{\geq\beta}$ (resp. $\mathcal{D}_{>\beta}$), is defined as $\mathcal{D}_{\geq\beta} = \{O_1 \cup O_2\} \cup \{t(\langle C, C', r, \alpha\rangle) : \langle C, C', r, \alpha\rangle \in \mathcal{M}, \alpha \geq \beta\}$ (resp. $\mathcal{D}_{>\beta} = \{O_1 \cup O_2\} \cup \{t(\langle C, C', r, \alpha\rangle) : \langle C, C', r, \alpha\rangle \in \mathcal{M}, \alpha > \beta\}$).*

The β-cut set of $\mathcal{D}$ is the union of O_1, O_2 and axioms translated from correspondences in the mapping whose confidence values are greater than or equal to β. It is adapted from the notion of cut set in possibilistic DLs in [QPJ07]. In Example 4, $\mathcal{D}_{>0.65} = O_1 \cup O_2 \cup \{t(m_3), t(m_4), t(m_5)\}$.

Definition 15 *Given a distributed system $\mathcal{D} = \langle O_1, O_2, \mathcal{M}\rangle$, the inconsistency degree of $\mathcal{D}$, denoted by $Inc(\mathcal{D})$, is defined as $Inc(\mathcal{D}) = max\{\alpha : \mathcal{D}_{\geq\alpha}$ is inconsistent$\}$.*

The inconsistency degree of a distributed system is the maximum confidence value α such that the α-cut set of $\mathcal{D}$ is inconsistent.

We describe the idea of our algorithm (Algorithm 3) as follows. Given a distributed system $\mathcal{D} = \langle O_1, O_2, \mathcal{M}\rangle$, we first translate the mapping $\mathcal{M}$ into a possibilistic DL knowledge base $O_{\mathcal{M}}$. Our algorithm first computes the *inconsistency degree* of the distributed system. Suppose $O_{\mathcal{M}} = \{(\phi_i, \alpha_i) : i = 1, ..., n\}$ where n is the number of correspondences. Let us rearrange the weights of axioms in $O_{\mathcal{M}}$ such that $\beta_0 > \beta_2 > ... > \beta_l > 0$, where β_i $(i = 0, ..., l)$ are all the distinct weights appearing in $O_{\mathcal{M}}$. Let $S_i = \{\phi : (\phi, \alpha) \in O_{\mathcal{M}}, \alpha = \beta_i\}$. Suppose $Inc(\mathcal{D}) = \beta_i$. We revise S_i by $\mathcal{D}_{>\beta_i}$[2]. Suppose the result of revision is $S_i' \cup \mathcal{D}_{>\beta_i}$. Let $\mathcal{M}' = \mathcal{M} \setminus \{\langle C, C', r, \alpha\rangle : \langle C, C', r, \alpha\rangle \in \mathcal{M}, \alpha = \beta_i, t(\langle C, C', r, \alpha\rangle) \in (S_i \setminus S_i')\}$. We then compute the

[1] A possibilistic DL knowledge base is a finite set of weighted DL axioms, where the weight of an axiom is interpreted as the *necessity degree* of the axiom.

[2] We do not specify a revision operator in our algorithm. However, the conditions that a revision operator should satisfy will be specified. To instantiate our algorithm, we will give a specific algorithm for DL knowledge base revision later.

Algorithm 3: A revision-based algorithm for repairing mappings

```
Data: A distributed system $\mathcal{D} = \langle O_1, O_2, \mathcal{M}\rangle$ and a revision operator $\circ$
Result: A repaired distributed system $\mathcal{D}_\circ = \langle O_1, O_2, \mathcal{M}_\circ\rangle$
begin
    $b := 0,\ d_{inc} := 0$
    $\mathcal{D}_\circ := \mathcal{D}$
    Rearrange the weights in $\mathcal{M}$ such that $\beta_0 > \beta_2 > ... > \beta_l > 0$
    $S_i := \{t(\langle C, C', r, \alpha\rangle) : \langle C, C', r, \alpha\rangle \in \mathcal{M}, \alpha = \beta_i, i = 0, ..., l\}$
    while $\mathcal{M}_\circ$ in $\mathcal{D}_\circ$ is inconsistent do
        $d_{inc} := GetInconsistencyDegree(\mathcal{D}_\circ, b)$
        if $\beta_k == d_{inc}$ then
            $S_{temp} := S_k \setminus (S_k \circ (\mathcal{D}_\circ)_{>d_{inc}})$
            $\mathcal{M}_\circ := \mathcal{M}_\circ \setminus \{\langle C, C', r, \alpha\rangle : t(\langle C, C', r, \alpha\rangle) \in S_{temp}, \alpha = \beta_k\}$
            $b := k$
    return $\mathcal{D}_\circ$
end
```

inconsistency degree of the distributed system $\mathcal{D}' = \langle O_1, O_2, \mathcal{M}'\rangle$. Suppose $Inc(\mathcal{D}') = \beta_j$ where $j > i$. We revise S_j by $\mathcal{D}'_{>\beta_j}$. We iterate the revision process until the mapping becomes consistent.

In line 2 of Algorithm 3, we set the initial value of variable b which will be used in the procedure $GetInconsistencyDegree(\mathcal{D}_\circ, b)$, where b is the begin pointer used by the binary search in $GetInconsistencyDegree(\mathcal{D}_\circ, b)$. The procedure $GetInconsistencyDegree(\mathcal{D}_\circ)$ is similar to the algorithm for computing the inconsistency degree in [QPJ07].

We have not specified a revision operator in Algorithm 3. However, we require that the revision operator $\circ$ used in the algorithm should satisfy the following properties:

- Inclusion: $O \circ O' \subseteq O \cup O'$
- Success: $O' \subseteq O \circ O'$

That is, the revision operator should result in an ontology which is a subset of the union of the original ontology and the newly received ontology and is a superset of the newly received ontology. These two conditions can infer that for any ontologies O and O' in a DL language L, $O \circ O'$ is an ontology in language L.

In the following, we present an algorithm to implement a concrete revision operator which is similar to the relevance-based debugging algorithm given in Chapter 3. We handle unsatisfiable concepts in the union of the local ontologies and the ontology translated from the mapping one by one until we resolve the inconsistency. For each unsatisfiable concept to be handled, we first select axioms that are relevant to it iteratively by the relevance-based selection function until the concept is unsatisfiable in these axioms. We find a local hitting set for the selected subontologies by calling the procedure COMPUTE_JUSTS(C, O, O'). We then select those axioms that are directly relevant to the selected axioms and further expand the local hitting set. We continue this process until the inconsistency is resolved.

Example 7 *To illustrate Algorithm 3, we follow Example 4. First of all, we need to reorder all the distinct confidence values in a descending order $\beta_0 = 0.93 > \beta_1 = 0.8 > \beta_2 = 0.65$ and the corresponding layers of correspondence axioms are $S_0 = \{t(m_5)\}$, $S_1 = \{t(m_3), t(m_4)\}$ and $S_2 = \{t(m_1, m_2)\}$ respectively. Then, we go into the while loop in line 6 since $\mathcal{M}$ is inconsistent. Based on the current $\mathcal{D}_\circ$ and $b = 0$, we obtain the current inconsistency degree 0.8 by Algorithm 4 (see* ***Part 1*** *for details). So $k = 1$. As we know that $\beta_1 = 0.8$, we use $((D)_\circ)_{>0.8}$ to revise S_1 and the revision result is $(S_k \setminus \{t(m_3)\}) \cup (\mathcal{D}_\circ)_{>0.8}$ according to Algorithm 2 (see* ***Part 2*** *for details). Therefore, we remove m_3 from $\mathcal{M}_\circ$ (see lines 9 and 10). Then we assign the current $k = 1$ to the begin pointer b and go to another iteration of while loop. Since the modified*

Algorithm 4: GetInconsistencyDegree

```
Data: A distributed system $\mathcal{D}_\circ = \langle O_1, O_2, \mathcal{M} \rangle$ and the begin pointer $b_0$
Result: The inconsistency degree $d_{inc}$
begin
    $b := b_0$
    $\mathcal{D} := \mathcal{D}_\circ$
    $m := 0$        // $m$ is the middle pointer of the binary search
    Rearrange the weights in $\mathcal{M}$ such that $\beta_0 > \beta_1 > ... > \beta_l > 0$
    $e := l$
    if $\mathcal{D}_{\geq \beta_e}$ is consistent then
        return 0
    else if $\mathcal{D}_{\geq \beta_b}$ is inconsistent then
        return $\beta_b$
    else
        while $b < e$ do
            $m := \ulcorner (b + e)/2 \urcorner$
            if $\mathcal{D}_{\geq \beta_m}$ is consistent then
                $b := m$
            else
                $e := m$
            if $e = b + 1$ then
                return $\beta_m$
end
```

$\mathcal{M}_\circ$ becomes consistent when m_3 is removed from it, the whole process of Algorithm 3 can be terminated and the result is $\mathcal{D}_\circ = \langle O_1, O_2, \mathcal{M} \setminus \{m_3\} \rangle$.

Part 1 (compute inconsistency degree): *The input data is $\mathcal{D}_\circ = \mathcal{D}$ and $b_0 = 0$. So $\mathcal{D} = \mathcal{D}_\circ$ and $b = 0$. The end pointer is $l = 2$ since we only have three distinct weights in $\mathcal{M}$. Since $\mathcal{D}_{\geq \beta_2}$ is inconsistent and $\mathcal{D}_{\geq \beta_0}$ is consistent, we go to the while loop in line 12 of Algorithm 4. Since $b = 0 < e = 2$, we go into while loop and get $m = 1$. Since $\mathcal{D}_{\geq \beta_m}$ is inconsistent, we assign $m = 1$ to e. According to lines 18 and 19, we return $\beta_1 = 0.8$ as the result.*

Part 2 (relevance-revision): *The input data is $O = S_1$ and $O' = \mathcal{D}_{>0.8}$. Suppose the first found unsatisfiable concept is $article$. We keep on selecting the k-relevant axioms in $O \cup O'$ w.r.t. the concept $article$ until $O_t = O \cup O'$ (i.e. $article$ becomes unsatisfiable in O_t). Then we go to line 16 to compute the minimal conflict sets and a diagnose (see **Part 2-1**). We get the conflict set $\{t(m_3)\}$ and a diagnose $HS = \{t(m_3)\}$. After this, we go to lines 17 and 18 of Algorithm 5. Since all the axioms have been selected, we can terminate the process and return $(S_1 \setminus \{t(m_3)\}) \cup \mathcal{D}_{>0.8}$.*

Part 2-1 (compute justifications): *The input data is $C = article$, $O = S_1$ and $O' = \mathcal{D}_{>0.8}$ for Algorithm 6. First of all, we compute a minimal conflict set $J = \{t(m_3)\}$ in line 2. Since there is only one axiom in J, we get $HS = \{t(m_3)\}$ in line 4. In the while loop, we return $(\{\{t(m_3)\}\}, \{t(m_3)\}$ since $O \setminus \{t(m_3)\} \cup O'$ becomes consistent (see lines 6 and 7).*

Algorithm 5: REL_REVISION(O, O')

```
Data: Two ontologies $O$ and $O'$
Result: A revised ontology $O \circ O'$
begin
    Global: $\mathcal{J} \leftarrow \emptyset$
    $k \leftarrow 1$
    $O_t \leftarrow HS \leftarrow \emptyset$
    forall $C \in$ GETALLCONCEPTS$(O \cup O')$ do
        if $O \cup O' \models C \sqsubseteq \bot$ then
            $\mathcal{S}_{rel} \leftarrow s_k(O \cup O', C)$
            while $\mathcal{S}_{rel} \neq \emptyset$ do
                $O_t \leftarrow O_t \cup \mathcal{S}_{rel}$
                if $HS \neq \emptyset$ then
                    if $C$ is satisfiable in $O \cup O' \setminus HS$ then
                        break
                    $(\mathcal{J}, HS') \leftarrow$ COMPUTE_JUSTS$(C, O_t \cap O \setminus HS, O_t \cap O')$
                    $HS \leftarrow HS \cup HS'$
                else if $C$ is unsatisfiable in $O_t$ then
                    $(\mathcal{J}, HS) \leftarrow$ COMPUTE_JUSTS$(C, O_t \cap O, O_t \cap O')$
                $k \leftarrow k + 1$
                $\mathcal{S}_{rel} \leftarrow s_k(O \cup O', C)$
            return $(O \setminus HS) \cup O'$
end
```

Algorithm 6: COMPUTE_JUSTS(C,O,O')

```
Data: Two ontologies $O$ and $O'$, and an unsatisfiable concept $C$ of $O \cup O'$
Result: A set of MUPS $\mathcal{J}$ for $C$ in $O$ and a hitting set $HS$
begin
    $J \leftarrow$ SINGLE_JUST$(C, O, O')$
    $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
    $HS = \{ax\}$ for some $ax \in J$
    while true do
        if $C$ is satisfiable in $(O \setminus HS) \cup O'$ then        /* Local hitting set */
            return $(\mathcal{J}, HS)$
        $J \leftarrow$ SINGLE_JUST$(C, O \setminus HS, O')$
        $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
        $HS = \{ax\}$ for some $ax \in J$
end
```

Chapter 5

Experiments

In this section, we present the evaluation results of our algorithms. The algorithms were implemented in Java as part of the RaDON plugin[1] for the NeOn Toolkit [2] using KAON2 as a reasoner. To fairly compare with the debugging algorithm in [KPHS07] and mapping repairing algorithm in [MST07], we re-implemented the algorithms with KAON2 API (we call them as All_Just_Alg algorithm and RepairMapping algorithm respectively).

The experiments regarding debugging were performed on a Linux 2.6.16.1 System and 1024MB maximal heap space was set. Sun's Java 1.5.0 Update 6 was used for Java-based tools. For computing justifications of a single unsatisfiable concept in each run, we set a time limit of 30 minutes. As the data sets for repairing mappings are relatively small, our experiments about repairing were performed on a laptop with 1.69 GHz Intel Pentium-M processor and 1 GB of RAM using Windows XP Service Pack 2. Sun's Java 1.5.0 Update 6 was used for Java-based tools and the maximum heap space was set to 800MB.

5.1 Data Sets

5.1.1 Data sets for debugging

Ontology	Classes	Properties	EquClass	SubClass.	DisjClass.	SubProp.	Domain	Range	Axioms
CHEM-A	48	19	18	46	6	4	18	18	114
CONFTOOL-CMT (HMatch)	68	95	1	111	70	0	95	95	457
CRS-SIGKDD (HMatch)	64	45	7	81	12	0	44	44	213
Proton	266	111	0	278	1,346	49	82	60	1,826
CONFTOOL-EKAW (HMatch)	112	69	0	189	117	8	60	60	485
EKAW-SIGKDD (HMatch)	123	61	7	204	74	8	51	51	440
KM1500	9,842	548	0	8,853	2,091	0	548	548	12,656

Table 5.1: Statistics of the ontologies in the data set.

Table 5.1 shows some characteristics of the data sets used for our experiments. *CHEM-A* was provided by Maryland University[3]. *Proton* is a basic upper-level ontology enriched with disjointness axioms[**?**]. *KM1500* is created by the Text2Onto ontology learning tool. The rest of data sets[4] are generated by merging ontologies and axioms resulting from a translation of the matching-results of ontology alignment system HMatch [**?**]. All the data sets are available for download[5].

We divide the data sets into two groups. Group one consists of four ontologies: *CHEM-A*, *CONFTOOL-CMT*, *CRS-SIGKDD*, *Proton*. For these ontologies, all the justifications of an unsatisfiable concept can be computed within the resource limits (i.e. the maximal heap space is 1024MB and the maximal time-out period is 30

[1] http://radon.ontoware.org/
[2] http://www.neon-toolkit.org/
[3] http://www.mindswap.org/ontologies/debugging/
[4] http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html
[5] http://radon.ontoware.org/downloads/datasets-iswc08.zip

minutes). Group two consists of three ontologies: *CONFTOOL-EKAW*, *EKAW-SIGKDD*, *KM1500*. For some unsatisfiable concepts in these ontologies, we cannot compute all of the justifications within the resource limits. Particularly, for ontology *KM1500*, it is impossible to compute all the justifications for most of the unsatisfiable concepts. Still, we use the ontologies to show how relevant justifications can be computed.

5.1.2 Data sets for mapping repair

Source ontology	Target ontology	# Axioms (source / target)	# Mappings	# Unsati. concepts
CRS	EKAW	69 / 248	44	42
EKAW	CMT	248 / 246	46	18
EKAW	CRS	248 / 69	80	47
EKAW	SIGKDD	248 / 122	70	42

Table 5.2: Statistics of data sets for mapping repair.

For this scenario, we use the ontology mapping data sets provided by the University of Mannheim.[6] The data sets include some source ontologies and mappings used in the ontology alignment evaluation initiative[7], which provides a platform to evaluate ontology matching systems. For our test, we use as source ontologies different ontologies about the domain of scientific conferences: *CONFTOOL* (a $\mathcal{SIF}(\mathcal{D})$ ontology), *CMT* (a $\mathcal{ALCIF}(\mathcal{D})$ ontology), *EKAW* (a $\mathcal{SHIN}$ ontology), *CRS* (a DL-Lite ontology) and *SIGKDD* (a $\mathcal{ALI}(\mathcal{D})$ ontology). The pairwise mappings were generated automatically by the HMatch system. We selected these mappings for our experiments because they are more problematic and the resulting ontology including source ontology, target ontology and their mappings contains more unsatisfiable concepts. Thus it would be more interesting to deal with those harder parts of mappings.

5.2 Experimental Results on Relevance-based Debugging Algorithm

ontology	Strategy	# Unsatisf. Concepts	# Justifications All	# Justifications Avg	Justification_Size Avg	# Hitting sets Avg
CHEM-A	All_Just_Alg	37	412	11	9	195
	All_Just_Relevance	37	412	11	9	195
	CM_Just_Relevance	37	37	1	7	4
CONFTOOL-CMT	All_Just_Alg	26	351	14	6	191
	All_Just_Relevance	26	351	14	6	191
	CM_Just_Relevance	26	43	2	4	3
CRS-SIGKDD	All_Just_Alg	19	64	3	5	14
	All_Just_Relevance	19	64	3	5	14
	CM_Just_Relevance	19	21	1	4	3
Proton	All_Just_Alg	24	41	2	6	9
	All_Just_Relevance	24	41	2	6	9
	CM_Just_Relevance	24	24	1	6	4

Table 5.3: The evaluation results for data sets in group one.

To evaluate the efficiency of our algorithm over data sets in group one, we compute all the justifications for all unsatisfiable concepts using All_Just_Alg and our relevance-based one with both strategies (i.e. All_Just_Relevance and CM_Just_Relevance). Figure 5.1 shows the average execution time for each unsatisfiable concept.[8] We observe that our algorithm is much faster than All_Just_Alg to compute the justifications in average when strategy CM_Just_Relevance is chosen. This is because fewer justifications are generated. Take the ontology *CONFTOOL-EKAW* as an example. The average number of justifications returned by All_Just_Relevance is about 14, whilst the average number is 2 when strategy CM_Just_Relevance is chosen. Another observation is that our algorithm outperforms All_Just_Alg when strategy All_Just_Relevance

[6]http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html
[7]http://om2006.ontologymatching.org/
[8]Please note that the results show the total time to compute the justifications, including the time to check satisfiability, unlike the results reported in [KPHS07], which excluded the time for satisfiability checking.

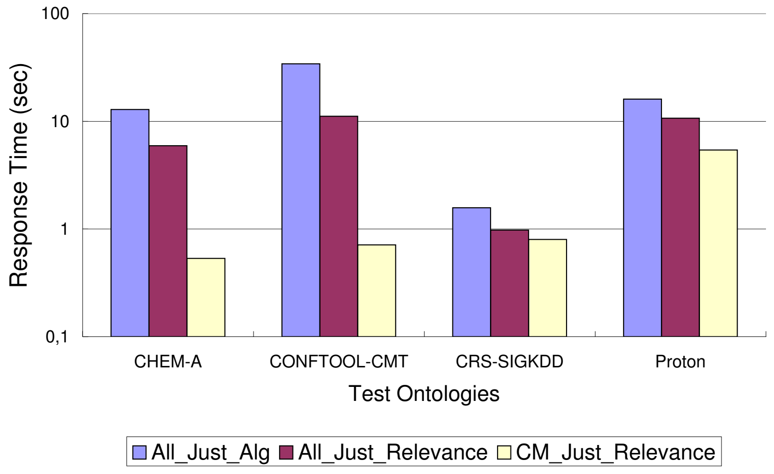

Figure 5.1: The average time to compute justifications for an unsatisfiable concept.

is chosen. It shows the advantage of incrementally increasing the size of the ontology using the selection function.

Relevance Degree k	1	2	3	4	5	6	7	8	9
s_{rel}(CONFTOOL-EKAW, Trip, k)	5	26	93	251	372	468	483	483	483
s_{rel}(EKAW-SIGKDD, Workshop_Paper, k)	5	59	173	275	368	427	435	435	435
s_{rel}(KM1500, framework, k)	57	1,335	6,678	9,730	10,892	11,206	11,305	11,336	11,342
s_{rel}(KM1500, experience, k)	48	1,286	6,675	9,856	10,897	11,218	11,307	11,335	11,342

Table 5.4: Number of axioms selected by s_{rel} depending on the relevance degree k

Table 5.4 shows the number of axioms that have been selected by the selection function s_{rel} for a given relevance degree k. We can see that sometimes the set of axioms is expanded very fast using the relevance-based selection function. For example, $|s_{rel}(O, C, 1)| = 57$, $|s_{rel}(O, C, 2)| = 1,278$ and $|s_{rel}(O, C, 3)| = 5,343$ for C=framework and O is the ontology KM1500. However, by considering the large size of ontology KM1500 (i.e. 12,656 axioms in total), it is still promising to apply the relevance-based selection function to alleviate the heavy burden to find justifications over the entire ontology. The advantage of using relevance-based algorithm can be further seen in the results below. We show the evaluation results

(O, C)	Strategy	# of Justifications $k=1$	$k=2$	$k=3$	Total # of found justifications	# Found hitting sets	Time (sec)
(CONFTOOL-EKAW, Trip)	All_Just_Alg	n.a.	n.a.	n.a.	34	2,208	TO
	All_Just_Relevance	0	0	80	80	2	TO
	CM_Just_Relevance	0	0	3	3	2	3
(EKAW-SIGKDD, Workshop_Paper)	All_Just_Alg	n.a.	n.a.	n.a.	41	3,279	TO
	All_Just_Relevance	0	61	TO	61	20	TO
	CM_Just_Relevance	0	4	–	4	2	2
(KM1500, framework)	All_Just_Alg	n.a.	n.a.	n.a.	1	0	TO
	All_Just_Relevance	1	33	TO	34	2	TO
	CM_Just_Relevance	1	9	–	10	2	96
(KM1500, experience)	All_Just_Alg	n.a.	n.a.	n.a.	1	0	TO
	All_Just_Relevance	1	6	10	17	6	TO
	CM_Just_Relevance	1	–	–	1	1	6

Table 5.5: The evaluation results over some unsatisfiable concepts.

for some unsatisfiable concepts in the data sets of group two in Table 5.5, where “TO" means time-out (the time-out period is set to be 30 minutes) and “n.a." means “not applicable". “–" indicates the algorithm has terminated.. The table shows the number of justifications for a given unsatisfiable concept depending on the relevance degree k (note that this does not apply to All_Just_Alg), the total number of found justifications and hitting sets, and the computation time. If the timeout was exceeded, the total number of justifications and hitting sets refers to the number found until the timeout occurred. Strategy CM_Just_Relevance performs very fast for all the selected unsatisfiable concepts. We can make the following observations: (1) For the large data set *KM1500* with more than 10 thousand axioms, only one justification is found by the All_Just_Alg within 30 minutes. This is because this algorithm performs each satisfiability check over the entire ontology, which is quite time-consuming. In contrast, our relevance-based algorithm only performs satisfiability check over relatively smaller sub-ontologies $k_{rel}(O, C, k)$ when $k \leq 3$. Take the concept “experience" as an example. Our algorithm with input strategy All_Just_Relevance returns 1, 6 and 10 justifications for $k = 1$ ($|O'| = 48$), $k = 2$ ($|O'| = 1,286$) and $k = 1$ ($|O'| = 6,675$) respectively. (2) For data sets CONFTOOL-EKAW and EKAW-SIGKDD, our algorithm with input strategy All_Just_Relevance returns more justifications than All_Just_Alg within the time limit and these justifications are most relevant to the unsatisfiable concept. (1) and (2) together show the advantage of introducing the selection function to find justifications incrementally. (3) Our algorithm with input strategy CM_Just_Relevance is much more efficient than other two (e.g. 3 seconds for concept “Trip" in *CONFTOO-EKAW*, 96 seconds for concept “framework" in *KM1500*), although much less justifications are returned.

5.3 Experimental Results on Revision-based Algorithm for Mapping Repair

In the following, we evaluate our algorithm with respect to two measures: efficiency and meaningfulness.

Evaluation Measures To measure the efficiency, we provide the revision time including the time to check whether the ontology is incoherent as well as the time to resolve the incoherence.
We also consider the measure of meaningfulness in [QHH$^+$08]. Two users were asked to assess whether the removal of an axiom in a particular revision was correct from their point of view. We provided the correspondences in a mapping which have been removed by the algorithms to resolve incoherence. For each correspondence, we asked the users to decide whether the removal: (1) was correct, (2) was incorrect, or (3) whether they are unsure. For the evaluated results returned by each user, the correctness is then measured by the ratio of correct removals:

$$\text{Correctness} = \frac{\#Correct\_Removals}{\#Total\_Removals}$$

Similarly we can define an “Error_Rate” based on the incorrect removals and an “Unknown_Rate” based on the removals where the users were unsure. We combine the obtained Correctness (respectively Error_Rate and Unknown_Rate) values from different users by averaging them. Note that the correctness is the same as the measure of repair precision in [MST07].

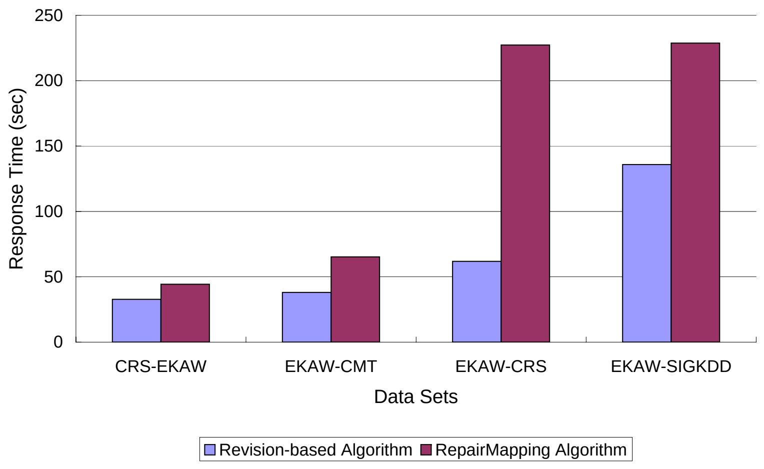

Figure 5.2: The performance of mapping revision algorithms.

Evaluation Results We compare our algorithm with the algorithm given in [MST07] by simulating their algorithm[9], which we call RepairMapping algorithm. Figure 5.2 shows the efficiency of two algorithms to repair mappings. Obviously, our revision-based algorithm outperforms RepairMapping algorithm in [MST07] according to the selected data sets. Especially, for data set *EKAW-CRS*, the time for RepairMapping algorithm is about four times as much as that for our algorithm. This is because, for our algorithm, (1)the search space is limited by stratifying the correspondences according to their associated confidence values and thus only a subset of all correspondences need to be checked for removal; (2)with stratification technique and selection function, each satisfiability checking has been done based on a subset of a resulting ontology O_r which includes a source ontology, target ontology and the correspondences between them. However for RepairMapping algorithm, the search space involves all the correspondences and every satisfiability checking is based on the entire ontology O_r.

Data Set	Algorithm	Correctness	Error_Rate	Unknown_Rate
CRS-EKAW	Revision-based Algorithm	0.67	0.10	0.24
	RepairMapping Algorithm	0.60	0.15	0.25
EKAW-CMT	Revision-based Algorithm	0.93	0	0.07
	RepairMapping Algorithm	0.88	0	0.12
EKAW-CRS	Revision-based Algorithm	0.78	0.20	0.03
	RepairMapping Algorithm	0.65	0.30	0.05
EKAW-SIGKDD	Revision-based Algorithm	0.97	0.03	0
	RepairMapping Algorithm	0.89	0.11	0

Table 5.6: Meaningfulness of the algorithms to repair mapping.

Table 5.6 shows the results for the meaningfulness of the mapping repair based on the expert users' assessment whether the removal was correct. That is, if the definition of a removed axiom does not make sense according to the expert users' experience, we consider the removal as correct.

From Table 5.6 we can see that overall the rate of correct removals for our algorithm is considerable higher than that of the erroneous removals. This shows that generally the ranking of axioms in our approach works well for resolving incoherence. By comparing with the meaningfulness of the RepairMapping algorithm, our algorithm shows higher rate of correct removal and lower rate of error removal or unknown removal in most cases. This observation is what we have expected. RepairMapping algorithm randomly computes a minimal conflict set and then chooses an axiom with the lowest confidence value in this conflict set to remove when resolving incoherence. Whilst our algorithm removes an axiom not only when it belongs to a minimal conflict set and has lowest weight in this minimal conflict set but also when this minimal conflict set has not be repaired by removing those axioms whose weights are greater than the lowest weight in it. For example, there are two minimal conflict sets $\mathcal{C}_1 = \{m_1, m_3\}$ and $\mathcal{C}_2 = \{m_3, m_5\}$ for concept $article$ in Example 4. Since a RepairMapping algorithm randomly computes a minimal conflict set, we assume that it compute $\mathcal{C}_1$ first. The the algorithm removes m_1. Since the revised mapping is still inconsistent. The algorithm computes another minimal conflict set $\mathcal{C}_2$ and removes m_3. Then the mapping becomes consistent. Whilst for our algorithm, we will compute $\mathcal{C}_2$ first because m_3 has higher weight than m_1 and m_5. Therefore, our algorithm removes m_3 but it does not remove m_1.

[9]The algorithm given in [MST07] is based on distributed description logics [BS02], where mappings are translated into sets of directed correspondences. So they use different semantics to interpret mappings in networked ontologies than ours.

Chapter 6

Conclusion

6.1 Summary

In this deliverable we proposed a relevance-based algorithm to find MUPS for unsatisfiable concepts and a revision-based algorithm to repair inconsistent mappings in networked ontologies. These two algorithms together will provide important support for diagnosing and repairing inconsistent networked ontologies.

For the relevance-based algorithm, we first introduced a relevance-based ordering on MUPS for an unsatisfiable concept and thus provided a criterion of comparison between different MUPS. We then provided a novel algorithm to find MUPS for an unsatisfiable concept based on a relevance-based selection function. Our algorithm allows for two different strategies when calculating MUPS: All_Just_Relevance and CM_Just_Relevance. Using the first strategy, our algorithm can find all the MUPS for an unsatisfiable concept. When the second strategy is chosen, our algorithm usually does not calculate all the MUPS but a subset of them that satisfies some minimality condition. Based on experimental results, we showed that our algorithm is very promising compared with the HST-based algorithm in [KPHS07] for both strategies. More specifically, our algorithm is much more efficient than the HST-based one when the strategy CM_Just_Relevance is chosen. Although only a small subset of the MUPS are returned by our algorithm when the strategy CM_Just_Relevance is chosen, these MUPS still provide partially complete view of the unsatisfiability as they correspond to hitting sets for the set of all the MUPS.

For the revision-based algorithm, we adapted the revision-based algorithm in possibilistic logic given in [Qi08] to find a diagnosis for the distributed system by treating $\mathcal{M}$ as a possibilistic DL knowledge base. In our algorithm, we need to compute the inconsistency degree of a distributed system and we do not specify a concrete revision operator. We adapted the algorithm for computing the inconsistency degree in possibilistic DL knowledge base given in [QPJ07]. We also presented an algorithm to implement a concrete revision operator which is similar to the relevance-based debugging algorithm given in Chapter 3. Our algorithm has been implemented by using KAON2 reasoner. Our evaluation on the algorithm for repairing mappings was done by considering the following evaluation measures: the first measure is the runtime of the algorithm and the second measure is the correctness or meaningfulness of the results of our approach. By our experimental results, we can see that our algorithm can handle real life networked ontologies and results in meaningful repairs to the mappings in the distributed system. We also compared our algorithm with an algorithm that simulates the algorithm for repairing mappings given in [MST07]. The results showed that our algorithm is more efficient and effective than the simulated algorithm.

6.2 Roadmap

There are a couple of problems left for future work. First, in the evaluation of our relevance-based algorithm, while we have shown that even a simple syntax-based selection function yields useful results, for future work we will investigate more powerful selection functions, such as a semantic relevance selection function.

We will also extend our algorithm by considering uncertainty. Second, we consider only two measures for evaluation of our revision-based algorithm. As a future work, we will compare our algorithm with the simulated algorithm by considering some other measures, such as those given in [MST07].

Bibliography

[BCM$^+$03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[BPS07] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic EL$^+$. In *Proc. of KI'07*, pages 52–67, 2007.

[BS02] Alexander Borgida and Luciano Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *CoopIS/DOA/ODBASE*, pages 36–53, 2002.

[dlBSW03] Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*, pages 32–43, 2003.

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

[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.

[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.

[HvHtT05] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *Proc. of 19th International Joint Conference on Artificial Intelligence(IJCAI'05)*, pages 254–259. Morgan Kaufmann, 2005.

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *Proc. of ISWC/ASWC'07*, pages 267–280, 2007.

[KPSH05] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.

[MS07] Christian Meilicke and Heiner Stuckenschmidt. Applying logical constraints to ontology matching. In *Proc. of KI'07*, pages 99–113, 2007.

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

[MST08] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Reasoning support for mapping revision. *Journal of Logic and Computation, to appear*, 2008.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In *Proc. of WWW'05*, pages 633–640, 2005.

[QHH$^+$08] Guilin Qi, Peter Haase, Zhisheng Huang, Qiu Ji, Jeff Z. Pan, and Johanna Völker. A kernel revision operator to support ontology evolution. In *Proc. of ISWC'08, to appear*. 2008.

[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.

[QHJV07] Guilin Qi, Peter Haase, Qiu Ji, and Johanna Voelker. D1.2.2 consistency models for networked ontologies–evaluation. Technical Report D1.2.2, Universität Karlsruhe, September 2007.

[Qi08] Guilin Qi. A semantic approach for iterated revision in possibilistic logic. In *AAAI'08*, pages 523–527. 2008.

[QPJ07] Guilin Qi, Jeff Z. Pan, and Qiu Ji. Extending description logics with uncertainty reasoning in possibilistic logic. In *Proc. of ECSQARU'07*, pages 828–839, 2007.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of IJCAI'03*, pages 355–362, 2003.

[Sch05] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *Proc. of ESWC'05*, pages 226–240, 2005.

[SHCvH07] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *J. Autom. Reasoning*, 39(3):317–349, 2007.

[ST05] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005*, pages 361–376, 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.