NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D1.2.2 Consistency Models for Networked Ontologies — Evaluation

Deliverable Co-ordinator: **Guilin Qi, Peter Haase**

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

Other Authors: **Qiu Ji, Johanna Voelker**

In this deliverable, we report on evaluations of approaches to handling (in)consistencies in networked ontologies. In particular, we evaluate our general framework for resolving inconsistency and incoherence introduced in NeOn deliverable D1.2.1. We first provide two specific approaches for resolving incoherence and one specific approach for resolving inconsistency. The evaluations are performed against various data sets from NeOn case studies, which exhibit different forms of networking relationships. Our evaluation is done by considering the following evaluation measures: the first measure is the characterization of inconsistency, the second measure is the runtime of the algorithm and the third measure is the correctness or meaningfulness of the results of our approach.

Document Identifier:	NEON/2007/D1.2.2/v1.0	Date due:	August 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2007
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

Change Log

Version	Date	Amended by	Changes
0.1	19-06-2007	Guilin Qi	Create the deliverable
0.5	08-08-2007	Qiu Ji	Results of Experiments
0.9	13-08-2007	Peter Haase	Final Write-up
0.95	05-09-2007	Guilin Qi, Peter Haase, Qiu Ji	Addressing reviewers comments
1.0	07-09-2007	Peter Haase	Final QA

Executive Summary

In this deliverable, we evaluate our general framework for resolving inconsistency and incoherence introduced in NeOn deliverable D1.2.1. We first provide two specific approaches for resolving incoherence and one specific approach for resolving inconsistency. We give two algorithms to repair an incoherent ontology. The first algorithm is based on a score ordering and the second one is based on certainty degrees attached to terminology axioms. We also give an algorithm to find one cardinality-maximal consistent subset of the ABox w.r.t. the TBox and another ontology. The algorithms are implemented in a software component RaDON for *Reasoning and Diagnosis of Ontology Networks*.

The evaluations are performed against various real life data sets (including those from NeOn case studies). The ontologies are fairly large and vary in size and complexity. The ontologies exhibit various forms of logical contradictions, in particular incoherences and inconsistencies, both on the TBox and ABox level. Further, we consider networked ontologies in particular, where inconsistency comes from mapping between ontologies, as well as ontologies that consist of a number of individual modules.

In our evaluation we consider two important use cases in resolving inconsistencies: *diagnosis and repair*, where we start with given an inconsistent ontology that we need to diagnose and repair, and *consistent ontology evolution*, where we are starting out with a consistent ontology and incrementally add new axioms, which may lead to inconsistencies.

Our evaluation is done by considering the following evaluation measures: the first measure is the characterization of inconsistency, the second measure is the runtime of the algorithm and the third measure is the correctness or meaningfulness of the results of our approach.

Overall, the results show that in principle our algorithms can be applied for real-life use cases on real-life data.

In particular we observe that

- Our approach is able to identify and resolve various types of inconsistencies occurring in the data sets.
- The algorithms perform on ontologies with several thousands of axioms.
- The automated algorithms results in meaningful repairs to the ontology.

In the future, we intend to deploy our RaDON component in the NeOn case study prototypes, in particular integrated into the ontology learning process of the FAO case study, in which inconsistencies are most likely to occur. The integration of RaDON as an engineering component into the NeOn toolkit is already ongoing.

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

Recently, 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. Current DL reasoners, such as RACER [HM01] and FaCT [Hor98], can detect

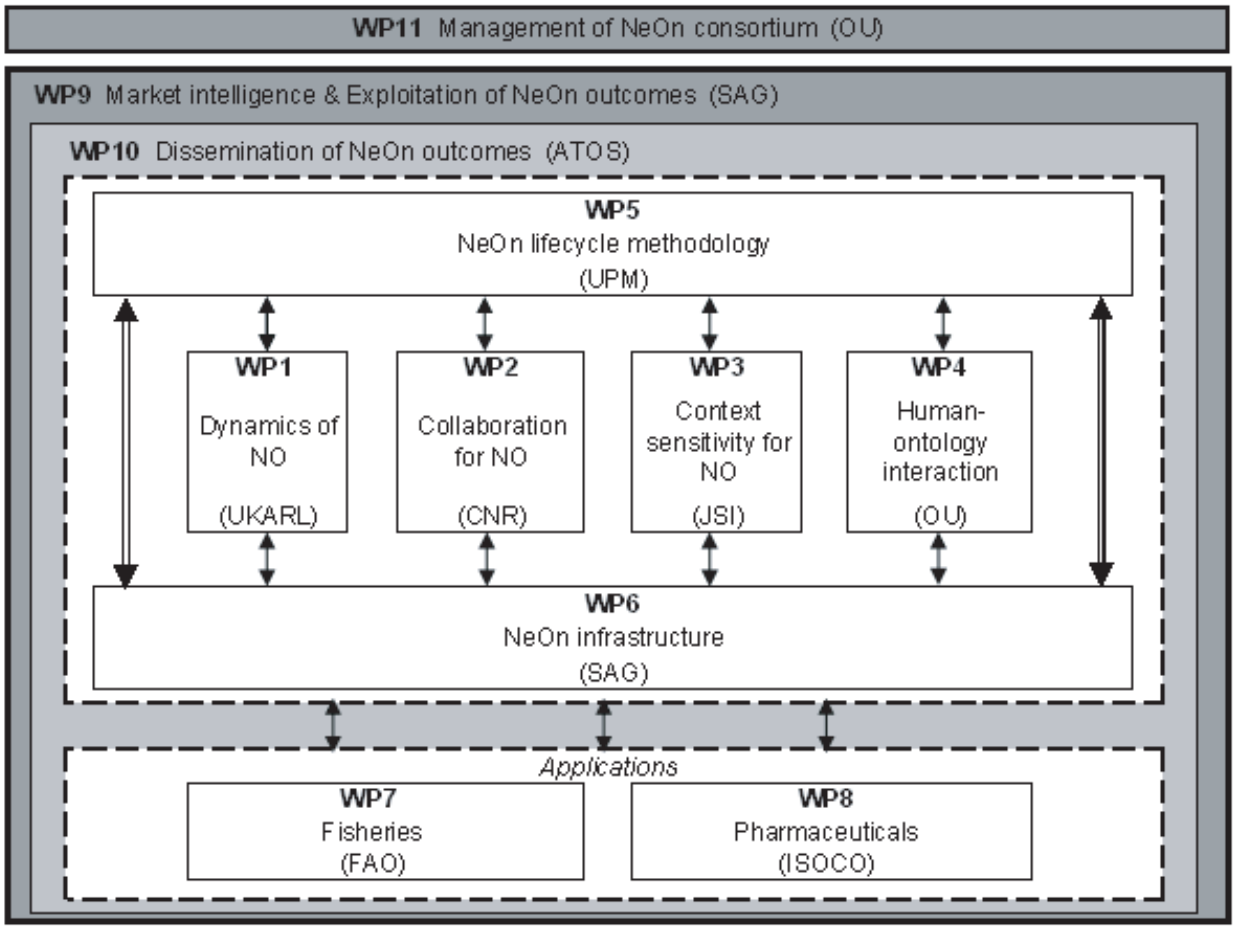


Figure 1.1: Relationships between different workpackages in NeOn

logical inconsistency. However, they only provide lists of unsatisfiable classes. The process of *resolving* inconsistency is left to the user or ontology engineers. The need to improve DL reasoners to reason with inconsistency is becoming urgent to make them more applicable. Many approaches were proposed to handle inconsistency in ontologies based on existing techniques for inconsistency management in traditional logics, such as propositional logic and nonmonotonic logics [PSK05, HvHH$^+$05, Sch05, SC03, FPA05, HvHtT05].

There are mainly two ways to deal with inconsistent ontologies [HvHtT05]. One way is to simply avoid the inconsistency and to apply a non-standard reasoning method to obtain meaningful answers. A general framework for reasoning with inconsistent ontologies based on *concept relevance* was proposed in [HvHtT05]. The idea is to select from an inconsistent ontology some consistent sub-theories based on a *selection function*, which is defined on the syntactic or semantic relevance. Then standard reasoning on the selected sub-theories is applied to find *meaningful* answers. The second way to deal with logical contradictions is to resolve logical modeling errors whenever a logical problem is encountered. In [MLB05], the authors proposed an algorithm for inconsistency handling by transforming every GCI in a DL knowledge base into a cardinality restriction, and a cardinality restriction responsible for a conflict is weakened by relaxing the restrictions on the number of elements it may have. In [QLB06b], a revision-based algorithm for handling inconsistency in description logics is proposed which generalizes the approach in [MLB05]. Several methods have been proposed to debug erroneous terminologies and have them repaired when inconsistencies are detected [SC03, Sch05, PSK05, FS05].

In deliverable D1.2.1 [QHJ07] we have presented a general framework of consistency models for networked ontologies considering the second way of resolving inconsistency. In this deliverable we further detail this

framework and additionally provide evaluations for our approach.
The main goals of this evaluation are

- to show that our approach is able to handle the various forms of inconsistencies that we encounter in dealing with real-life ontologies
- to assess the scalability of the algorithms and their implementation for real-life ontologies
- to show the meaningfulness of the results obtained by our approach.

In order to evaluate these aspects, we define a number of corresponding evaluation measures.
An important objective of our evaluation is to operate on real life-data. We then select a number of different data sets that are obtained from actual NeOn case study data, or from scenarios similar to those found in NeOn case studies. We also use data sets that are *networked*, in particular, we have one data set consisting of a large number of modules, where the modules are related via ontology mappings.

1.3 Overview of the Deliverable

This deliverable is structured as follows. In Chapter 2, we provide some key preliminaries to understand our notions of inconsistency and incoherence handling. In Chapter 3 we recapitulate the main ideas of our approach to resolving inconsistencies as presented in deliverable [QHJ07]. We then provide a specific instantiation our general approach in Chapter 4. In Chapter 5 we describe the evaluation settings, discussing the evaluation methodology, data sets, evaluation measures, In Chapter 6 we present the evaluation results from our experiments. We conclude with a summary and a roadmap for future work in Chapter 7.

Chapter 2

Preliminaries

We assume that the reader is familiar with Description Logics (DLs) and refer to Chapter 2 of the DL handbook [BCM$^+$03] for an excellent introduction. A DL knowledge base (or local ontology) O consists of a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. A TBox contains intensional knowledge such as concept definitions of the form $C \sqsubseteq D$, where C and D are concept. An ABox contains extensional knowledge and is used to describe individuals. Throughout the deliverable, let $\mathcal{T} = \{Ax_1, ..., Ax_n\}$ be a set of (terminological) axioms, where Ax_i is of the form $C_i \sqsubseteq D_i$ for each $1 \leq i \leq n$ and arbitrary concepts C_i and D_i.

Different notions of inconsistency in DLs have been used in the Semantic Web community. In [SC03], Schlobach et al. proposed an approach to debug inconsistent ontologies, in which inconsistency is identified with the existence of unsatisfiable concepts. In [HvHtT05], Huang et al. developed a framework of reasoning with inconsistent ontologies, in which inconsistency is given a classical first-order logic interpretation. In [FHP$^+$06], the distinction between inconsistency and incoherence is introduced to provide a general foundation for inconsistency processing in DL-based ontologies.

The first type of problem is the unsatisfiability of a single concept in a terminology, i.e., a TBox of an ontology O.

Definition 1 (Unsatisfiable Concept) *A named concept C in a terminology $\mathcal{T}$, is unsatisfiable iff, for each model $\mathcal{I}$ of $\mathcal{T}$, $C^{\mathcal{I}} = \emptyset$.*

That would lead us to consider the kinds of terminologies and ontologies with unsatisfiable concepts.

Definition 2 (Incoherent Terminology) *A terminology $\mathcal{T}$ is incoherent iff there exists an unsatisfiable named concept in $\mathcal{T}$.*

Definition 3 (Incoherent Ontology) *An ontology O is incoherent iff its terminology is incoherent.*

Current DL reasoners, such as RACER, can detect logical incoherence and return unsatisfiable concepts in OWL ontologie. However, they do not support the diagnosis and incoherence resolution at all. To explain logical incoherence, it is important to debug *relevant* axioms which are responsible for the contradiction.

Definition 4 *[SC03] Let A be a named concept which is unsatisfiable in a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a* minimal unsatisfiability-preserving sub-TBox (MUPS) *of $\mathcal{T}$ if A is unsatisfiable in $\mathcal{T}'$, and A is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$. The set of all MUPS of $\mathcal{T}$ w.r.t A is denoted as $MU_A(\mathcal{T})$*

A MUPS of $\mathcal{T}$ $w.r.t$ A is the minimal sub-TBox of $\mathcal{T}$ in which A is unsatisfiable. We will abbreviate the set of MUPS of $\mathcal{T}$ $w.r.t$ a concept name A by $mups(\mathcal{T}, A)$. Let us consider an example from [SC03]. Suppose $\mathcal{T}$ contains the following axioms:

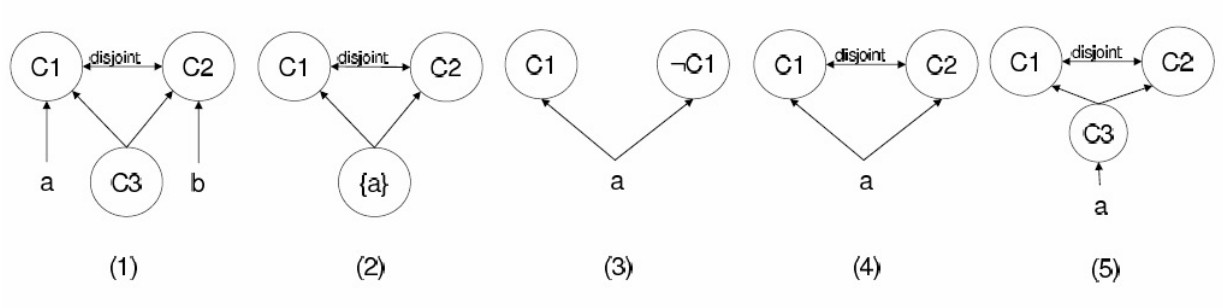


Figure 2.1: Examples of variants of inconsistency and incoherence.

$$ax_1 : A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3 \qquad ax_2 : A_2 \sqsubseteq A \sqcap A_4$$
$$ax_3 : A_3 \sqsubseteq A_4 \sqcap A_5 \qquad ax_4 : A_4 \sqsubseteq \forall s.B \sqcap C$$
$$ax_5 : A_5 \sqsubseteq \exists s.\neg B \qquad ax_6 : A_6 \sqsubseteq A_1 \sqcup \exists r.(A_3 \sqcap \neg C \sqcap A_4)$$
$$ax_7 : A_7 \sqsubseteq A_4 \sqcap \exists s.\neg B$$

where A, B and C are atomic concept names and A_i $(i = 1, ..., 7)$ are defined concept names, and r and s are atomic roles. In this example, the unsatisfiable concept names are A_1, A_3, A_6, A_7 and MUPS of $\mathcal{T}$ w.r.t A_i $(i = 1, 3, 6, 7)$ are:
$mups(\mathcal{T}, A_1) : \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$
$mups(\mathcal{T}, A_3) : \{ax_3, ax_4, ax_5\}$
$mups(\mathcal{T}, A_6) : \{\{ax_1, ax_2, ax_4, ax_6\}, \{ax_1, ax_3, ax_4, ax_5, ax_6\}\}$
$mups(\mathcal{T}, A_7) : \{ax_4, ax_7\}$
MUPS are useful for relating sets of axioms to the unsatisfiability of specific concepts, but they can also be used to calculate a minimal incoherence preserving sub-TBox, which relates sets of axioms to the incoherence of a TBox in general and is defined as follows.

Definition 5 *[SC03] Let $\mathcal{T}$ be an incoherent TBox. A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a* minimal incoherence-preserving sub-TBox (MIPS) *of $\mathcal{T}$ if $\mathcal{T}'$ is incoherent, and every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent. The set of all MIPSs of $\mathcal{T}$ is denoted as $MI(\mathcal{T})$.*

A MIPS of $\mathcal{T}$ is the minimal sub-TBox of $\mathcal{T}$ which is incoherent. The set of MIPS for a TBox $\mathcal{T}$ is abbreviated with $mips(\mathcal{T})$. For $\mathcal{T}$ in the above example, we get 3 MIPS:
$mips(\mathcal{T}) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$
However, incoherence does not provide the classical sense of the inconsistency because there might exist a model for an incoherent ontology. Thus, the classical inconsistency for ontologies is introduced.

Definition 6 (Inconsistent Ontology) *An ontology O is inconsistent iff it has no model.*

Further, we say that an ontology $O = \langle \mathcal{T}, \mathcal{A} \rangle$ is TBox-inconsistent, iff the TBox alone (without considering the ABox) has no model, and it is ABox-inconsistent, iff the ABox alone (without considering the TBox) has no model.

The relationship between incoherence and inconsistency is not simple. Firstly, the fact that an ontology is inconsistent does not necessarily imply that it is incoherent, and vice versa. There exist different combinations of inconsistency and incoherence, as illustrated in Figure 2.1 and discussed in the following.

Figure 2.1 (1) is an example of a consistent but incoherent ontology, in which the two disjoint concepts $C1$ and $C2$ share a sub-concept $C3$. Figure 2.1 (2)-(4) show examples of inconsistent ontologies. Figure 2.1 (2) is an

example of an inconsistent but coherent ontology, in which the two disjoint concepts $C1$ and $C2$ share a subconcept which is a nominal $\{a\}$. Figure 2.1 (3) is an example of an inconsistent ontology in which an instance a instantiates a concept $C1$ and its complement $\neg C1$. Figure2.1 (4) is an example of an inconsistent but coherent ontology, in which the two disjoint concepts $C1$ and $C2$ share an instance a. Finally, Figure2.1 (5) shows an example of an ontology that is both incoherent and inconsistent.

From the definitions of incoherence above, we know that incoherence can occur in the terminology level only. When dealing with inconsistency, we can differentiate terminology axioms and assertional axioms. We have the following categorization of different kinds of reason for inconsistent ontologies.

- *Inconsistency due to terminology axioms*: In this case, we only consider inconsistency in TBoxes. Figure 2.1 (2) is an example of such an inconsistency. Following our definitions, this kind of inconsistency will make the TBox incoherent.
- *Inconsistency due to assertional axioms:* This kind of inconsistency only occurs in ABoxes. It is not related to incoherence. A source of assertional inconsistency is that there are conflicting assertions about one individual, e.g., an individual is asserted to belongs to a class and its complement class, as in Figure 2.1 (2).
- *Inconsistency due to terminology and assertional axioms:* In this case, each conflicting set of axioms must contain both terminology axioms and assertional axioms. This kind of inconsistency is sometimes dependent on incoherence. Such an example is shown in Figure 2.1 (5). It is easy to see that C_3 is an unsatisfiable concept and that O is inconsistent. The reason for the inconsistency is that the individual a instantiates C_3 which is unsatisfiable. Therefore, if we repair the unsatisfiable concept C_3, then the inconsistency will disappear as well. On the other hand, the inconsistency in the example in Figure 2.1 (4) is not caused by an incoherence.

The first kind of inconsistency is only related to terminology axioms. In this case, the unit of change is a concept (either atomic or complex). Therefore, some revision approaches which take the individual names as the unit of change, such the one proposed in [QLB06a], cannot be applied to deal with this kind of inconsistency. By contrast, the other two kinds of inconsistency are related to assertional axioms. So the unit of change can be either a concept or an individual name.

From this discussion we observe that the causes for incoherence and inconsistency are manifold and their interdependencies are complex. Incoherence is always caused by conflicts in the terminology. It may or may not affect the consistency of the overall ontology. Inconsistency may arise due to conflicts in the ABox, in the TBox, or a combination of both ABox and TBox. In the following section we propose a revision approach resolving conflicts in evolving ontologies that takes these interdependencies into account.

Chapter 3

A General Approach for Resolving Inconsistency and Incoherence

In this chapter, we deal with the problem of resolving inconsistency and incoherence. More specifically, we consider this problem in the context of ontology evolution. In this case, this problem is similar to the belief revision in classical logic.

The problem of revision of ontology is described as follows. Suppose we have two ontologies $O = \langle \mathcal{T}, \mathcal{A} \rangle$ and $O' = \langle \mathcal{T}', \mathcal{A}' \rangle$ where O is the original ontology and O' is the newly received ontology which contains a set of axioms to be added to O. Even if both O and O' are individually consistent and coherent, putting them together may cause inconsistency or incoherence. Therefore, we need to delete or weaken some axioms in O to restore consistency and coherence. Note that when the original ontology is inconsistent or incoherent and the newly received ontology is empty, then the problem of revision of ontology is reduced to the problem of resolving inconsistency and incoherence in a single ontology. Therefore, the approach proposed in this chapter can be also used to deal with a single ontology.

Usually, the result of revision is a set of ontologies rather than a unique ontology [QLB06a]. More formally, we have the following definition of ontology revision. We denote all the possible ontologies with $\mathcal{O}$.

We first introduce the notion of a disjunctive ontology from [MLB05]. A disjunctive ontology, denoted as **O**, is a set of ontologies. The semantics of the disjunctive ontology is defined as follows [MLB05]:

Definition 7 *A disjunctive ontology* $\mathbf{O}$ *is satisfied by an interpretation* $\mathcal{I}$ *(or* $\mathcal{I}$ *is a model of* $\mathbf{O}$*) iff* $\exists O \in \mathbf{O}$ *such that* $\mathcal{I} \models O$*.* $\mathbf{O}$ *entails* ϕ*, denoted* $\mathbf{O} \models \phi$*, iff every model of* $\mathbf{O}$ *is a model of* ϕ*.*

Definition 8 *An ontology revision operator (or revision operator for short) in DLs is a function* $\circ : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{P}(\mathcal{O})$ *which satisfies the following conditions: 1)* $O \circ O' \models \phi$ *for all* $\phi \in O'$*, where* $\mathcal{P}(\mathcal{O})$ *denotes all the subsets of* $\mathcal{O}$*; 2) for each* $O_i \in O \circ O'$*,* O_i *is consistent.*

That is, an ontology revision operator is a function which maps a pair of ontologies to a disjunctive ontology which can consistently infer the newly received ontology. In practice, we may only need one ontology after revision. In this case, we can obtain such an ontology by ranking the ontologies obtained by the revision operator and then selecting the one with highest rank. Ranking of ontologies can either be given by the users or be computed by some algorithms.

The current work on ontology revision suffers from some problems, to name a few, we have the following ones:

- There is much work on the analysis of applicability of AGM postulates for belief change to DLs [FPA05, FHP$^+$06]. However, few of them discuss the concrete construction of a revision approach.
- Current revision approaches often focus on dealing with logical inconsistency. Another problem which is as important as inconsistency handling is incoherence handling, where an ontology is incoherent if

and only if there is an unsatisfiable named concept in its terminology. As analyzed in [FHP+06], logical incoherence and logical inconsistency are not independent of each other. A revision approach which resolves both logical incoherence and inconsistency is missing.

We now propose our general approach which resolves incoherence and inconsistency in an integrated way. The approach consists of the process steps shown in Figure 3.1. In this process, problems that are related only with either the TBox or the ABox are dealt with independently in two separate threads (c.f. left and right thread of Figure 3.1, respectively). For the TBox, inconsistency resolution is done before incoherence resolution because incoherence is a consequence of inconsistency in the TBox. We first check if $\mathcal{T} \cup \mathcal{T}'$ is consistent. If it is not, then we resolve inconsistency. This can be done by either deleting the erroneous terminology axioms or weakening them. In a next step, we resolve incoherence. There are several ways to resolve incoherence. The commonly used technique is to remove some (usually minimal numbers) of erroneous terminology axioms which are responsible for the incoherence. Alternatively, we can take the maximal coherent sub-ontologies of $\mathcal{T}$ w.r.t $\mathcal{T}'$ as the result of revision [MLBP06, LPSV06]. For the ABox, we resolve inconsistencies that occur only due to assertional axioms. This can be done by either deleting the assertional axioms which are responsible for the inconsistency or weakening them. The weakening of assertional axioms may be different from that of terminology axioms. Finally, we deal with the inconsistency due to both terminology and assertional axioms. In each of the revision steps, the result may be a disjunctive ontology, since there may exist several alternative way to resolve the incoherence or inconsistency. However, in each step a decision is made, which single ontology should be selected as input for the subsequent step. This decision can be made either by the user or an automated procedure based on a ranking of the results as discussed above.

Our general approach does not yet specify *how* to deal with inconsistency or incoherence. Moreover, for different kinds of inconsistency, we can use different strategies to resolve them. For example, when resolving inconsistency due to terminology axioms, we can take the maximal consistent subsets of the original TBox $w.r.t$ the new TBox as the result of revision. Whilst when resolving inconsistency related to assertional axioms, we can apply the revision approach in [QLB06a], which removes minimal number of individual resulting in the conflict. In the next section, we instantiate our approach by proposing a concrete approach to resolving incoherence and some concrete approaches to resolving inconsistency.

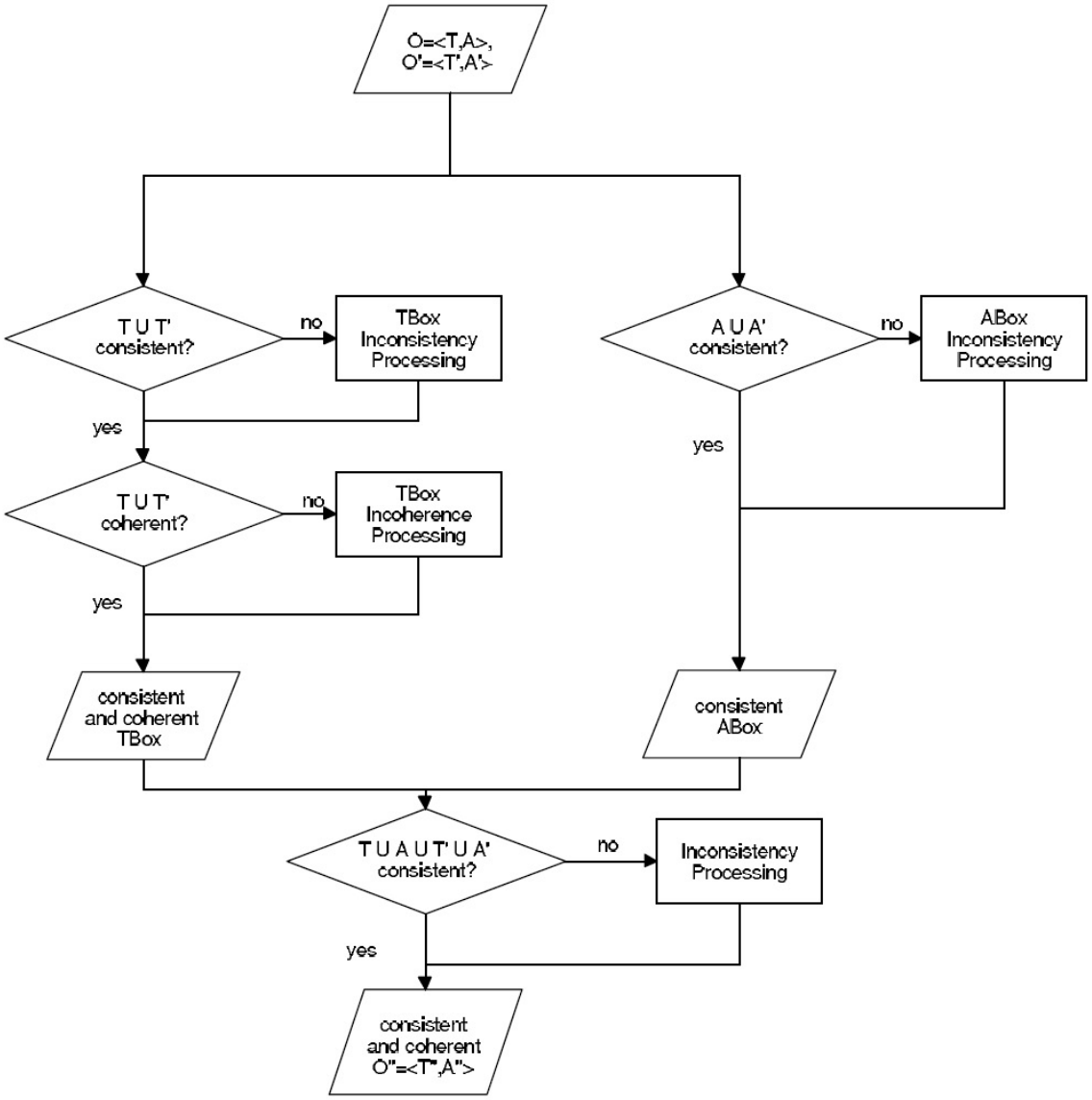


Figure 3.1: Approach to resolving inconsistency and incoherence

Chapter 4

Instantiation of the General Approach

When resolving incoherence or inconsistency, we need to delete or weaken some axioms in the original ontology. In this case, we want to keep as much information in the original ontology as possible. There are several ways to achieve this. For example, to resolve incoherence, we may need to delete minimal number of axioms to get a coherent ontology or we take a maximal consistent sub-ontology as the result of revision. In the following, we propose some concrete approaches to resolving incoherence and resolving inconsistency which capture some kinds of minimal change. When resolving incoherence or inconsistency in a TBox, the unit of change can be either an axiom or a concept. However, if there are axioms in ABox which are responsible for inconsistency, the unit of change can be either an individual, a concept, or an axiom.

4.1 Resolving incoherence

In this section, we propose an approach to resolving incoherence in ontology evolution. Resolving incoherence is a problem which is often overlooked during ontology evolution. This problem is easy to be confused with resolving inconsistency. Many debugging approaches have been proposed to pinpoint the unsatisfiable concepts or terminology axioms which are responsible for incoherence. There are mainly two ways to resolve the logical incoherence. The first way is to remove some (usually minimal numbers) of erroneous terminology axioms which are responsible for the incoherence to restore coherence. Alternatively, we can take the maximal coherent sub-ontologies of $O\ w.r.t\ O'$ as the result of revision [MLBP06]. Our approach to resolving incoherence belongs to the first class, i.e. we delete some terminology axioms to restore coherence.

We first generalize the concept of *minimal incoherence-preserving sub-TBox* (MIPS) defined in [SC03].

Definition 9 *Let $\mathcal{T}$ and $\mathcal{T}_0$ be two TBoxes, where $\mathcal{T}$ is an old TBox and $\mathcal{T}_0$ is a newly received TBox. A minimal incoherence-preserving sub-TBoxes (MIPS) $\mathcal{T}'$ of $\mathcal{T}\ w.r.t\ \mathcal{T}_0$ is a sub-Tbox of $\mathcal{T}$ which satisfies (1) $\mathcal{T}' \cup \mathcal{T}_0$ is incoherent; (2) $\forall \mathcal{T}'' \subset \mathcal{T}'$, $\mathcal{T}'' \cup \mathcal{T}_0$ is coherent. We denote the set of all MIPS of $\mathcal{T}\ w.r.t\ \mathcal{T}_0$ by $MIPS_{\mathcal{T}_0}(\mathcal{T})$.*

A MIPS of a TBox $\mathcal{T}\ w.r.t$ TBox $\mathcal{T}_0$ is the minimal sub-TBox of $\mathcal{T}$ that is incoherent with $\mathcal{T}_0$. It is similar to the *kernel* defined in [Han94]. In classical logic, given a knowledge base A which is a set of classical formulas and a formula ϕ, a ϕ-kernel of A is the minimal subbase of A that implies ϕ [Han94]. To define a contraction function called kernel contraction, Hansson defines an incision function which selects formulas to be discarded in each ϕ-kernel of A. We adapt the incision function to define our revision operator.

Definition 10 *Let $\mathcal{T}$ be a TBox. An* incision function *for $\mathcal{T}$, denoted as σ, is a function such that for each TBox $\mathcal{T}_0$*

(i) $\sigma(MIPS_{\mathcal{T}_0}(\mathcal{T})) \subseteq \bigcup_{\mathcal{T}_i \in MIPS_{\mathcal{T}_0}(\mathcal{T})} \mathcal{T}_i$;

(ii) if $\mathcal{T}' \in MIPS_{\mathcal{T}_0}(\mathcal{T})$ and $\mathcal{T}' \neq \emptyset$, then $\mathcal{T}' \cup \sigma(MIPS_{\mathcal{T}_0}(\mathcal{T})) \neq \emptyset$.

That is, an incision function for a TBox $\mathcal{T}$ is a function such that for each TBox $\mathcal{T}_0$, it selects formulas from every MIPS of $\mathcal{T}$ $w.r.t$ $\mathcal{T}_0$ if this MIPS is not empty. The incision function plays a similar role as concept pinpointing in [SC03].

An important incision function is the one which is called *minimal incision function* [FFKI06]. We redefine it as follows.

Definition 11 *Let $\mathcal{T}$ be a TBox. An incision function σ for $\mathcal{T}$ is* minimal *if there is no other incision function σ' such that there is a TBox $\mathcal{T}_0$, $\sigma'(MIPS_{\mathcal{T}_0}(\mathcal{T}))\subset\sigma(MIPS_{\mathcal{T}_0}(\mathcal{T}))$.*

We give a refined minimal incision function.

Definition 12 *Let $\mathcal{T}$ be a TBox. An incision function σ for $\mathcal{T}$ is* cardinality-minimal *if there is no other incision function σ' such that there is a TBox $\mathcal{T}_0$, $|\sigma'(MIPS_{\mathcal{T}_0}(\mathcal{T}))|< |\sigma(MIPS_{\mathcal{T}_0}(\mathcal{T}))|$.*

A cardinality-minimal incision function is always a minimal incision function.

From each incision function, we can define a revision operator.

Definition 13 *Let $\mathcal{T}$ be a TBox, and σ be an incision function for $\mathcal{T}$. The* kernel revision *operator $\circ_\sigma$ for $\mathcal{T}$ is defined as follows: for each TBox $\mathcal{T}_0$,*

$$\mathcal{T} \circ_s \mathcal{T}_0 = \{(\mathcal{T} \setminus \sigma(MIPS_{\mathcal{T}_0}(\mathcal{T}))) \cup \mathcal{T}_0\}.$$

The resulting TBox of the kernel revision operator only contains one TBox. For simplicity, we write $\mathcal{T} \circ_s \mathcal{T}_0 = (\mathcal{T} \setminus \sigma(MIPS_{\mathcal{T}_0}(\mathcal{T}))) \cup \mathcal{T}_0$ later. According to the definition of an incision function, the resulting TBox of the kernel revision operator is always a coherent TBox.

The kernel revision operator is defined by an incision function. There are many ways to give an incision function. In the following, we proposed an approach for computing an incision function by applying Reiter's hitting set algorithm [Rei87] .

Reiter introduced a hitting set tree algorithm to calculate diagnoses from conflict sets [Rei87]. Given a collection of sets $\mathcal{C}$, a *hitting set* for $\mathcal{C}$ is a set $H\subseteq \cup_{S\in\mathcal{C}} S$ such that $H\cap S\neq\emptyset$ for each $S\in\mathcal{C}$. A hitting set H for $\mathcal{C}$ is minimal if and only if the following conditions hold: (1) H is a hitting set; (2) for any $H'\in\mathcal{C}$, if $H\subset H'$, then H' is not a hitting set for $\mathcal{C}$. It has been shown in [SHC06] that a subset $\mathcal{T}'$ of $\mathcal{T}$ is a diagnosis for an incoherent TBox $\mathcal{T}$ if and only if $\mathcal{T}'$ is a minimal hitting set for the collection of conflict sets of $\mathcal{T}$. Unlike the approach in [SHC06], we use hitting set tree algorithm to calculate diagnoses from $MIPS_{\mathcal{T}_0}(\mathcal{T})$, which is the set of MIPSs of $\mathcal{T}$ w.r.t. $\mathcal{T}_0$. It is clear that each minimal hitting set of $MIPS_{\mathcal{T}_0}(\mathcal{T})$ defines a minimal incision function.

To calculate minimal hitting sets of $MIPS_{\mathcal{T}_0}(\mathcal{T})$, we can adapt Reiter's hitting set tree (HS-tree) algorithm and call the new algorithm as modified HS-tree algorithm, denoted as $ModifiedHST$. Given a collection $\mathcal{C}$ of sets, a HS-tree T is the smallest edge-labeled and node-labeled tree, such that the root is labeled by ✓ 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 ✓, $H(n)$, i.e. the labels of its path from the root, is a hitting set for $\mathcal{C}$. The HS-tree algorithm usually results in several minimal hitting sets. We randomly choose one of them as output of the algorithm.

Figure 4.1 shows a HS-tree 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).

Next, we give two algorithms to repair a TBox, depending on if the axioms in the TBox are attached to weights or not. The first algorithm is based on the scoring function on axioms which is defined as follows.

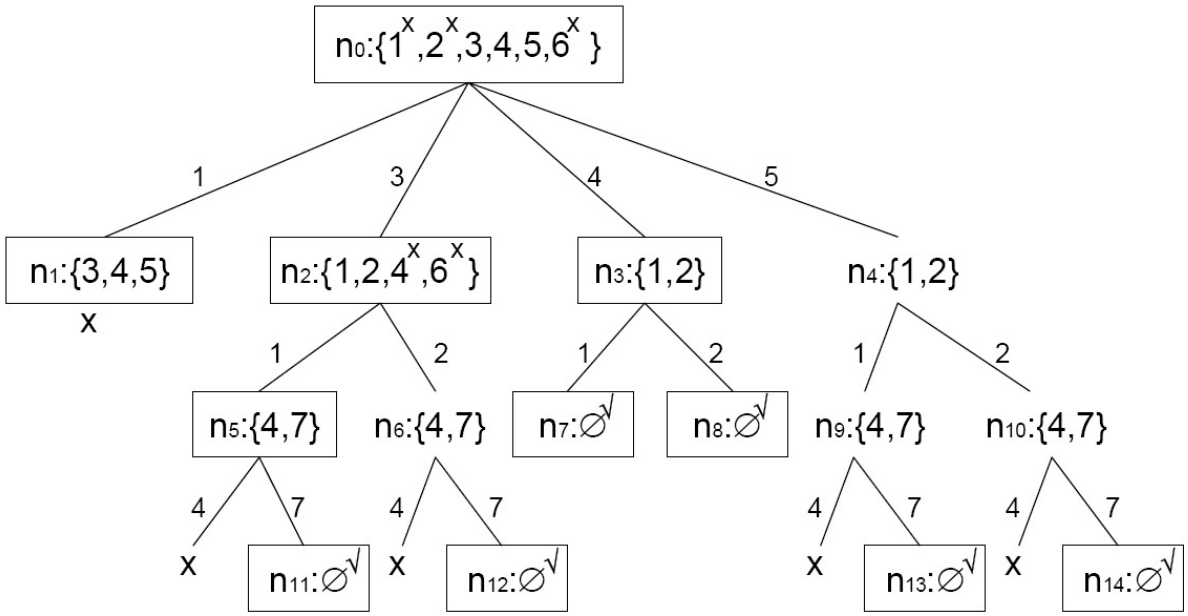


Figure 4.1: HS-Tree with small conflict sets

Definition 14 *Let $\mathcal{T}$ be a TBox. The scoring function for $\mathcal{T}$, is a function $S_{\mathcal{T}} : \mathcal{P}(\mathcal{T}) \mapsto N$ such that for all $\mathcal{T}' \in \mathcal{P}(\mathcal{T})$*

$$S_{\mathcal{T}}(\mathcal{T}') = |\{\mathcal{T}_i \in MI(\mathcal{T}) : \mathcal{T}_i \cap \mathcal{T}' \neq \emptyset\}|.$$

The scoring function $S_{\mathcal{T}}$ for $\mathcal{T}$ returns for each subset $\mathcal{T}'$ of $\mathcal{T}$ the number of MIPS of $\mathcal{T}$ that have an overlap with $\mathcal{T}'$. If we apply the scoring function to each singleton $\{ax_i\}$, where ax_i is an axiom in $\mathcal{T}$, then we can attached each axiom in $\mathcal{T}$ a degree.

Algorithm 1: Algorithm for Repair based on scoring function

Data: Two TBoxes $\mathcal{T}$ and $\mathcal{T}_0$, where $\mathcal{T}$ is the TBox to be revised
Result: A repaired coherent TBox $\mathcal{T} \circ_s \mathcal{T}_0$
begin
- $\mathcal{C} = \emptyset$
- **calculate** $MIPS_{\mathcal{T}_0}(\mathcal{T})$
- **for** $ax \in \bigcup_{\mathcal{T}_i \in MIPS_{\mathcal{T}_0}(\mathcal{T})} \mathcal{T}_i$ **do**
  - $w_{ax} := S_{MIPS_{\mathcal{T}_0}(\mathcal{T})}(\{ax\})$
- **for** $\mathcal{T}_i \in MIPS_{\mathcal{T}_0}(\mathcal{T})$ **do**
  - $A_i := \{ax \in \mathcal{T}_i : \nexists ax' \in \mathcal{T}_i, w_{ax'} > w_{ax}\}$
  - $\mathcal{C} := \mathcal{C} \cup \{A_i\}$
- $\sigma(MIPS_{\mathcal{T}_0}(\mathcal{T})) := ModifiedHST(\mathcal{C})$
- $\mathcal{T} \circ_s \mathcal{T}_0 := \mathcal{T} \setminus \sigma(MIPS_{\mathcal{T}_0}(\mathcal{T})) \cup \mathcal{T}_0$
- **return** $\mathcal{T} \circ_s \mathcal{T}_0$

end

In Algorithm 1, we first calculate all the MIPSs of $\mathcal{T}$ w.r.t. $\mathcal{T}_0$. The approach for calculating all the MIPSs is based on the CS-Tree algorithm for finding minimal unsatisfiable subsets in [dlBSW03] and HS-Tree algorithm in [KPGS06]. We refer the reader to the thesis [Sha07] for more details about our approach. We then compute the score of each axioms which are in the union of the MIPSs. For each MIPS, we select a subset

of it which contains those axioms whose scores are the minimal among all the axioms in the MIPS and apply the modified HST approach to them. The result of the modified HST approach is the set of axioms to be deleted. After removing these axioms, we restore coherence of the TBox $\mathcal{T}$ w.r.t. $\mathcal{T}_0$.

Example 1 *Suppose that we have two TBoxes :*

$\mathcal{T} = \{Example\_c \sqsubseteq Knowledge\_c, Document\_c \sqsubseteq \neg Knowledge\_c,$
$Form\_c \sqsubseteq Knowledge\_c, Firm\_c \sqsubseteq Organisation\_c\}$

and

$\mathcal{T}_0 = \{Document\_c \sqsubseteq Example\_c, Form\_c \sqsubseteq Document\_c,$
$Knowhow\_document\_c \sqsubseteq Document\_c, KnowledgeManagement\_service \sqsubseteq Service\}.$

The TBoxes are taken from a real life ontology $km1500\_i500$ *which is obtained by ontology learning. The MIPSs of* $\mathcal{T}$ *w.r.t.* $\mathcal{T}_0$ *are*

$\mathcal{T}' = \{Example\_c \sqsubseteq Knowledge\_c, Document\_c \sqsubseteq \neg Knowledge\_c\}$

and

$\mathcal{T}'' = \{Document\_c \sqsubseteq \neg Knowledge\_c, Form\_c \sqsubseteq Knowledge\_c\}.$

The score of the disjointness axiom $Document\_c \sqsubseteq \neg Knowledge\_c$ *is 2 because it belongs to both MIPSs. The scores of other axioms are 1. Therefore, we delete* $Document\_c \sqsubseteq \neg Knowledge\_c$ *and the result of revision is*

$\mathcal{T} \circ_s \mathcal{T}_0 = \{Example\_c \sqsubseteq Knowledge\_c,\ Form\_c \sqsubseteq Knowledge\_c,\ Firm\_c \sqsubseteq Organisation\_c,$
$Document\_c \sqsubseteq Example\_c, Form\_c \sqsubseteq Document\_c,$
$Knowhow\_document\_c \sqsubseteq Document\_c,\ KnowledgeManagement\_service \sqsubseteq Service\}.$

When axioms in the TBox are attached to confidence values, we do not need to calculate all the MIPSs before we repair the TBox. We have the following algorithm which utilizes confidence values of axioms to decide which axioms should be deleted. We use w_{ax} to denote the confidence value of the axiom ax.

Algorithm 2: Algorithm for Repair based on confidence values

Data: Two TBoxes $\mathcal{T}$ and $\mathcal{T}_0$, where $\mathcal{T}$ is the TBox to be revised, axioms in $\mathcal{T}$ are attached with confidence values

Result: A repaired coherent TBox $\mathcal{T} \circ_w \mathcal{T}_0$

begin

- $\mathcal{C} := \emptyset$
- **for** $C \in GETALLCONCEPTS(\mathcal{T} \cup \mathcal{T}_0)$ **do**
  - **while** $\mathcal{T} \cup \mathcal{T}_0 \models C \sqsubseteq \bot$ **do**
    - $\mathcal{M}_{C,\mathcal{T},\mathcal{T}_0} := GETMUPS_{\mathcal{T}_0}(C, \mathcal{T})$
    - **for** $\mathcal{T}_i \in \mathcal{M}_{C,\mathcal{T},\mathcal{T}_0}$ **do**
      - $\underline{\mathcal{T}_i} := \{ax \in \mathcal{T}_i : \nexists ax' \in \mathcal{T}_i, w_{ax'} < w_{ax}\}$
    - $\mathcal{C} := \{\underline{\mathcal{T}_i} : \mathcal{T}_i \in \mathcal{M}_{C,\mathcal{T},\mathcal{T}_0}\}$
    - $\mathcal{T}_C := ModifiedHST(\mathcal{C})$
    - $\mathcal{T} := \mathcal{T} \setminus \mathcal{T}_C$
    - $\mathcal{C} := \emptyset$
- **return** $\mathcal{T} \cup \mathcal{T}_0$

end

In Algorithm 2, when resolving incoherence of a TBox, we do not compute all the MUPSs and MIPSs. Instead, we resolve incoherence by iteratively dealing with unsatisfiable concepts. That is, we remove axioms in MUPSs of an unsatisfiable concept and make it satisfiable and then go to deal with another unsatisfiable concept, and so on. The algorithm which compute all the MUPSs of C in $\mathcal{T}$ w.r.t. $\mathcal{T}_0$ is similar to the algorithm to compute MUPS in [Sha07]. For each unsatisfiable concept, we take the subset of every MUPS

which contain axioms with minimal confidence values and apply the modified HST approach to select those axioms to be deleted.

Example 2 *Let us consider Example 1 again. Suppose axioms in the TBox $\mathcal{T}$ are attached with confidence values as follows: $w_{Example\_c \sqsubseteq Knowledge\_c} = 0.4$, $w_{Document\_c \sqsubseteq \neg Knowledge\_c} = 0.8$, $w_{Form\_c \sqsubseteq Knowledge\_c} = 0.6$, $w_{Firm\_c \sqsubseteq Organisation\_c} = 0.9$. There are two unsatisfiable concepts in $\mathcal{T} \cup \mathcal{T}_0$ are $Document\_c$ and $Form\_c$ respectively. Suppose our algorithm choose $Form\_c$ first. The MUPSs of $Form\_c$ in $\mathcal{T}$ w.r.t. $\mathcal{T}_0$ are $\mathcal{T}' = \{Document\_c \sqsubseteq \neg Knowledge\_c, Form\_c \sqsubseteq Knowledge\_c\}$ and $\mathcal{T}'' = \{Example\_c \sqsubseteq Knowledge\_c, Document\_c \sqsubseteq \neg Knowledge\_c\}$ respectively. Suppose the algorithm choose $\mathcal{T}'$ first. Since $w_{Form\_c \sqsubseteq Knowledge\_c} < w_{Document\_c \sqsubseteq \neg Knowledge\_c}$, we will delete $Form\_c \sqsubseteq Knowledge\_c$. $Form\_c$ is still unsatisfiable, so we choose $\mathcal{T}''$. Since $w_{Example\_c \sqsubseteq Knowledge\_c} < w_{Document\_c \sqsubseteq \neg Knowledge\_c}$, we will delete $Example\_c \sqsubseteq Knowledge\_c$. Let $\mathcal{T}_1 = \mathcal{T} \setminus \{Form\_c \sqsubseteq Knowledge\_c, Example\_c \sqsubseteq Knowledge\_c\}$. It is easy to check that $\mathcal{T}_1 \cup \mathcal{T}_0$ is coherent now. So the algorithm terminates and the result of revision is $\mathcal{T} \circ_w \mathcal{T}_0 = \{Document\_c \sqsubseteq \neg Knowledge\_c, Firm\_c \sqsubseteq Organisation\_c, Document\_c \sqsubseteq Example\_c, Form\_c \sqsubseteq Document\_c, Knowhow\_document\_c \sqsubseteq Document\_c, KnowledgeManagement\_service \sqsubseteq Service\}$.*

4.2 Resolving inconsistency

Many approaches for resolving inconsistency in classical logic are based on some maximal consistent subsets, e.g. the cardinality-maximizing revision approach in [Gin86]. The idea is that we select all the cardinality-maximizing subsets of the original knowledge base that are consistent with the new knowledge base. In this subsection, we apply the cardinality-maximizing revision approaches to dealing with inconsistency. We assume that axioms in the TBox is more reliable than those in ABox after incoherence step. Therefore, when there is an inconsistency, we will change the ABox only.

Definition 15 *Let $O = \langle \mathcal{T}, \mathcal{A} \rangle$ and $O' = \langle \mathcal{T}', \mathcal{A}' \rangle$ be two ontologies. A subset $\mathcal{A}_i$ of $\mathcal{A}$ is said to be cardinality-maximizing consistent $w.r.t$ O' and $\mathcal{T}$ if and only if it satisfies the following conditions:*

(m1) $\mathcal{A}_i \cup \mathcal{T} \cup \mathcal{T}' \cup \mathcal{A}' \not\models \bot$ and

(m2) $\forall \mathcal{A}_j \subseteq \mathcal{A}$, if $|\mathcal{A}_i| < |\mathcal{A}_j|$ then $\mathcal{A}_j \cup \mathcal{T} \cup \mathcal{T}' \cup \mathcal{A}' \models \bot$, where $|\mathcal{A}_i|$ is the cardinality of the set $\mathcal{A}_i$.

The set of all cardinality-maximal consistent sub-sets of $\mathcal{A}$ $w.r.t$ $\mathcal{T}$ and O' is denoted by $CMAXCON(\mathcal{A})_{\mathcal{T},O'}$.

Condition *m1* says that $\mathcal{A}_i$ is consistent with $\mathcal{T}$ and O', and Condition *m2* states that any sub-set of $\mathcal{A}$ that contains more elements than $\mathcal{A}_i$ is in inconsistent with $\mathcal{T}$ and O'.

We define a revision operator based on a cardinality-maximizing consistent sub-ABox.

Definition 16 *Let $O = \langle \mathcal{T}, \mathcal{A} \rangle$ and $O' = \langle \mathcal{T}', \mathcal{A}' \rangle$ be two ontologies. Then the cardinality-maximizing revision operator, denoted as $\circ_{CM}$, is defined as follows:*

$$O \circ_{CM} O' = \{\mathcal{A}_i \cup \mathcal{T} : \mathcal{A}_i \in CMAXCON(\mathcal{A})_{\mathcal{T},O'}\}.$$

According to the definition, the result of revision is a set of consistent ontologies. To select one from these ontologies, we need some extra information.

We present an algorithm 3 which returns a cardinality-maximizing consistent sub-ABox. In this algorithm, GetAllSubsets($\mathcal{T}$,$\mathcal{A}$,m) is to find all the subsets of $\mathcal{A}$ which satisfy that the cardinality of its union with $\mathcal{T}$ is m. We obtain the cardinality of the cardinality-maximizing consistent sub-ontologies of the original ontology O $w.r.t$ O' using a binary search (the while loop).

Algorithm 3: Algorithm to find a cardinality-maximizing consistent subABox

Data: An ontology $O = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is coherent
Result: A cardinality-maximizing consistent sub-ABox $\mathcal{A}_{cm}$ of $\mathcal{A}$
begin
  $\mathcal{S}$:=$\mathcal{T}$
  for *Axiom* $\in \mathcal{A}$ **do**
    $\mathcal{S}$.add(Axiom)
    if $\mathcal{S}$ *is inconsistent* **then**
      $\mathcal{S}$.remove(Axiom)
  b:=$|\mathcal{S}|$// b is the begin pointer of the binary search
  m:=0 //m is the middle pointer of the binary search
  e:=$|\mathcal{T}| + |\mathcal{A}|$ //e is the end pointer of the binary search
  while *b<e* **do**
    m:=$\lceil$(b+e)/2$\rceil$
    f:=false
    for *S* $\in$ *GetAllSubsets(*$\mathcal{T}$*,*$\mathcal{A}$*,m)* **do**
      if *S is consistent* **then**
        f:=true
        $\mathcal{A}_{cm}$:=S
        break
    if *f is false* **then**
      e:=m
    else
      b:=m
    if *m==e* **then**
      break
  return $\mathcal{A}_{cm}$
end

Chapter 5

Evaluation Settings

In this chapter, we discuss the evaluation settings for our experiments. In particular, we discuss the software implementation used for the evaluation, evaluation methodology, evaluation measures, and the data sets used for our experiments.

5.1 Implementation and Prototype

The algorithms described in the previous chapter have been implemented in our RaDON system for *Reasoning and Diagnosis of Ontology Networks*. In [QHJ07], we have already described the initial prototype of RaDON. RaDON is a system that extends the capabilities of existing reasoners with functionalities to deal with inconsistencies and incoherence. These additional functionalities are made accessible via extensions to the DIG interface. The idea of extending the DIG interface[1]with non-standard reasoning services has been developed originally in the SEKT project and has for example been applied in PION[2] and evOWLution[3]. The binary code of RaDON based on XDIG interface can be downloaded from the RaDON website `http://ontoware.org/projects/radon/`.
For the evaluation, we use an installation of the RaDON system on a Linux server running Sun's Java 1.5.0 Update 3 with a maximum heap space was set to 2048MB. For each test, the maximal time limit is 3 hours.

5.2 Evaluation Methodology

In our evaluations, we aim at addressing realistic real life scenarios with respect to the following aspects:

1. Realistic ontologies
2. Realistic size of data sets
3. Types of inconsistencies
4. Networking relationships
5. Realistic use cases

The first four aspects are considered by our selection of data sets. (c.f. Section 5.4 for a detailed description). In order to obtain realistic ontologies that are representative for the case studies, we opted to rely on ontologies from actual case studies created by ontology learning techniques. The ontologies are fairly large and vary in size and complexity. In addition, we also vary the sizes of the ontologies as an input parameter.

[1] http://dl.kr.org/dig/
[2] http://wasp.cs.vu.nl/sekt/pion/
[3] http://evowlution.ontoware.org/

The ontologies exhibit various forms of logical contradictions, in particular incoherences and inconsistencies, both on the TBox and ABox level. While most ontologies we use are single ontologies, we also consider networked ontologies, where inconsistency comes from mapping between ontologies, as well as ontologies that consist of a number of individual modules.

Further, in our evaluation we consider two different use cases in resolving inconsistencies:

- The first use case is that of *diagnosis and repair*. In this use case, we start with a given inconsistent ontology that we need to diagnose and repair.
- The second use case is that of *consistent ontology evolution*. In this use case, we are starting out with a consistent ontology and incrementally add new axioms, which may lead to inconsistencies.

Finally, we compare the effect of two different alternatives in repairing the ontologies:

- In the first alternative, we rely on a ranking of the axioms based on the scoring function, corresponding to Algorithm 1.
- In the second alternative, we rely on a ranking based on confidence values corresponding to Algorithm 2, provided by the ontology learning algorithms.

To evaluate the applicability of our approach in real life applications, we consider a number of evaluation measures that concern the aspects of *efficiency*, *effectiveness* and *scalability*, as we will discuss in the following.

5.3 Evaluation Measures

We consider three evaluation measures to test our system: the characteristics of inconsistency, the runtime of the algorithm, and the correctness of the results of the algorithm.

The first measure is the characteristics of inconsistency. This includes the following parameters: the type of inconsistency, i.e., is the testing ontology inconsistent or incoherent, the number of unsatisfiable concepts, the number of MIPS of the ontology. The time for debugging to find MUPSs and MIPSs is considered as well. Besides, the average size of one MUPS and the average size of one MIPS are measured. Ratio_Removed indicates the quotient by dividing the number of removed axioms to resolve incoherence by the number of distinct axioms in all MIPS :

$$\text{Avg\_MUPS\_Size} = \frac{\#Axioms\_in\_all\_MUPS}{\#MUPS}$$
$$\text{Avg\_MIPS\_Size} = \frac{\#Axioms\_in\_all\_MIPS}{\#MIPS}$$
$$\text{Ratio\_Removed} = \frac{\#Removed\_axioms}{\#Distinct\_axioms\_in\_all\_MIPS}$$

The second measure is the runtime of the algorithm, i.e. the time to identify and resolve an inconsistency. Specifically, the runtime for our experiments includes the time for debugging and repairing to make an incoherent or inconsistent ontology coherent and consistent. This is to test the scalability of the system. As indicated above, we evaluate the scalability for the different use cases.

Apart from scalability, we also evaluate the meaningfulness of the results of our system. We ask some users to evaluate the results of our algorithm. We provide several axioms which are selected to delete by our algorithm and the MIPSs and MUPSs containing them (and scores of the axioms or certainty degrees attached to the axioms if applicable). For each axiom, we ask the users to answer vote on three choices: make sense, not make sense and unknown. That is, the user can check the information about each axiom to be deleted and then either decide if the deletion of this axiom makes sense or does not make sense, or he or her does not know if this axiom should be deleted. The meaningfulness can be defined as the following, where #Removed_Axioms indicates the total number of removed axioms for a test ontology and #Removed_Axioms_MakeSense shows the number of removed axioms which make sense.

$$\text{Meaningfulness} = \frac{\#Removed\_Axioms\_MakeSense}{\#Removed\_Axioms}$$

5.4 Data Sets

In this section, we describe the ontologies that will be used for experiment.

Ontology	#TBox_Size	#ABox_Size	#Disjoint	#Modules	#Mappings	Incoherent	Inconsistent
proton_50_all	1,756	0	1,276	1	0	yes	no
proton_50_rudis	1,777	0	1,297	1	0	yes	no
proton_50_studis	1,826	0	1,346	1	0	yes	no
proton_100_all	1,096	0	616	1	0	yes	no
proton_100_rudis	1,411	0	931	1	0	yes	no
proton_100_studis	1,326	0	846	1	0	yes	no
km1500_i500	12,040	616	2,091	1	0	yes	yes
fao_species	2,969	536	517	1	0	yes	no
fao_glossaries	13,423	0	0	1,063	545,096	yes	no

Table 5.1: Statistics of Test Ontologies

1. LeDA: Proton

   In the context of 'Learning Disjointness' experiments, we created various sets of manually added disjointness axioms. As a basis for these data sets we chose a subset of PROTON ontology consisting of 266 classes, 77 object properties, 34 datatype properties and 1388 siblings. We then selected 2,000 pairs of classes each of which was randomly assigned to 6 different people - 3 PhD students from our institute (all of them are professional "ontologists") and 3 under-graduate students without profound knowledge in ontological engineering. Each of the annotators was given between 385 and 406 pairs along with natural language descriptions of the classes whenever those were available, the ontology and a definition of disjointness. Possible taggings for each pair were '$+$'(disjoint), '$-$' (not disjoint) and $?$ (unknown). Finally, we computed two majority votes for the different sets of annotated pairs of classes: If at least 50 percent (or 100 percent respectively) of the human annotators agreed upon '$+$' or '$-$' this decision was assumed to be the majority vote for that particular pair. We have the following 6 ontologies:

   - proton_50_all - all disjointness axioms, simple majority
   - proton_50_rudis - disjointness axioms created by experts, simple majority
   - proton_50_studis - disjointness axioms created by students, simple majority
   - proton_100_all - all disjointness axioms, perfect agreement
   - proton_100_rudis - disjointness axioms created by experts, perfect agreement
   - proton_100_studis - disjointness axioms created by students, perfect agreement

   All six ontologies are carefully engineered and the taxonomy can be assumed to be correct. However, some of the disjointness axioms may be incorrect, because they were added by different authors at a later stage of the engineering process. Not all of the authors necessarily took into account the taxonomy when modeling disjointness, and some of them (in particular, some of the students) possibly misunderstood the semantics of disjointness.

   The number of incorrect disjointness axioms in each of these ontologies certainly depends on the authors (experts or students) and their level of agreement (simple or complete majority). It is very likely that the least reliable dataset is 'proton_50_studis' whereas the most reliable one is 'proton_100_all'.

   In Table 5.1, the size of TBox and the number of disjointness axioms in proton ontologies are shown. Except the different number of disjointness axioms, these ontologies have the same number of other

kind of axioms. For example, all of them have 278 subclass axioms and 49 subproperty axioms. These ontologies are all incoherent but consistent.

The data set is interesting for evaluations, as the individual ontologies show a varying degree of incoherence.

2. Text2Onto: SEKT and KWeb

   The 'km1500_i500' ontology has been generated automatically from 1.500 abstracts of the 'knowledge management' information space which is part of the BT Digital Library. We applied the ontology learning framework Text2Onto in order to extract concepts, instances, taxonomic and non-taxonomic relationships, as well as disjointness axioms from these documents. All ontology elements are annotated with confidence (axioms, properties) or relevance (concepts, instances) values, e.g. based on lexical context similarity or the frequency of lexico-syntactic patterns matched in the text.

   The data set is interesting for evaluations, as it (1) contains both ABox and TBox axioms, (2) is large in size (this ontology has 12040 axioms in TBox and 616 axioms in ABox), and (3) the ontology is both inconsistent and incoherent.

3. Text2Onto: FAO Documents

   The 'fao_species' ontology was generated from 100 factsheets about different species of fish. It contains confidence and relevance values. The ontologies have been generated in a completely automatic manner. Therefore it is very likely that a very high number of axioms are incorrect.

   All the entities are annotated with relevance values, whereas all the axioms and properties have associated confidence values. However, it is unclear how reliable the confidence and relevance values are, and if they are truly correlated with the actual likeliness of an entity or axiom to be relevant or correct.

   This ontology is incoherent but consistent. It contains 2969 axioms in the TBox and 536 axioms in the ABox.

4. LExO: FAO Glossaries

   In order to generate this ontology, we first extracted a set of 1,063 natural language definitions from the FAO fishery glossary. These definitions were then transformed into OWL DL class descriptions by means of LExO resulting in a set of 1,063 modules with separate namespaces. We used FOAM for mapping these to the Agrovoc thesaurus, and finally obtained an ontology consisting of import statements and mapping axioms which allows for integrated reasoning over Agrovoc and the automatically generated class descriptions.

   The final data set contains 1,063 modules (each containing definitions for one glossary entry) as well as the mappings between each module and the Agrovoc thesaurus. The data set is thus interesting as an example of a networked ontology. We use this particular data set to evaluate the scalability of our approach with respect to the size of the ontology network. We do this by incrementally increasing the number of modules we consider. For each test, the mappings between each pair of modules we chose are included.

   Due to the provided mappings being between modules and a global ontology (i.e. the Agrovoc thesaurus), we extract automatically the mappings between each pair of modules from those existing ones. 187,013 pairs of modules have mappings and 545,096 mappings between modules are obtained. All these mappings are equivalent relations. Altogether, the TBox size of all modules is 13,423 and there is no ABox. This networked ontology is incoherent.

Chapter 6

Evaluation Results

6.1 The characterization of inconsistency

In this section, we take as input the entire ontologies in a subset of our data sets and output the characterization of inconsistency which has been described in Section 5.3.

Ontology	TBox_Size	#Disjoint_Axioms	#Unsatisfiable_Concepts	#MUPS	#MIPS	Avg_MUPS_Size	Avg_MIPS_Size	Debug_Time (s)
proton_50_all	1,756	1,276	11	21	10	5	4	2,252
proton_50_rudis	1,777	1,297	15	24	13	5	4	2,440
proton_50_studis	1,826	1,346	24	37	17	6	5	5,755
proton_100_all	1,096	616	3	3	3	5	5	110
proton_100_rudis	1,411	931	9	18	6	5	5	1,116
proton_100_studis	1,326	846	4	4	4	4	4	189
fao_species	2,969	517	9	10	9	3	3	3,649

Table 6.1: The characterization of inconsistency

In Table 6.1, TBox_Size means the number of axioms in TBox of the current ontology. #Disjoint indicates the number of disjoint-classes axioms. And the number of unsatisfiable concepts has been shown as #Unsatisfiable_Concepts. #MUPS and #MIPS describe the number of MUPS and MIPS respectively. Avg_MUPS_Size and Avg_MIPS_Size present the average size of a MUPS and MIPS separately. The last column gives the time to look for MUPS and MIPS.

For proton ontologies, we can see that the number of unsatisfiable concepts and MUPSs increase with the increase of the number of disjoint axioms. For example, 'proton_100_all' has the smallest number of disjoint axioms and thus it has the minimal extent of incoherence with 3 unsatisfiable concepts and 3 MUPSs and 3 MIPSs. For 'proton_50_studis', it has the largest number of disjoint axioms and then it is more noisier. The result here is consistent with what we have expected in Section 5.4.

For ontologies 'proton_50_all' or 'proton_50_rudis' with 'fao_species', although the two proton ontologies have more unsatisfiable concepts and MUPS than the latter, the debugging time for the latter is more due to the size of TBox. Although ontology 'fao_species' also has larger TBox than 'proton_50_studis', its debugging time is less than the latter because the latter has more MUPS. This shows that the debugging time is related not only to the size of TBox but also the number of MUPS.

Take some MUPS and MIPS from ontology proton_50_rudis as an example. One MUPS $MUPS_1$ for unsatisfiable concept "Employee" is given as following :

$MUPS_1 = \{Employee \sqsubseteq JobPosition, Employee \sqsubseteq \neg JobPosition\}$

The MUPS $MUPS_2$ is a MUPS of unsatisfiable concept "Manager" :

$MUPS_2 = \{Manager \sqsubseteq Employee, Employee \sqsubseteq JobPosition, Employee \sqsubseteq \neg JobPosition\}$

According to the two MUPS above, we can obtain one MIPS since $MUPS_1$ is included by $MUPS_2$:

$MIPS = \{Employee \sqsubseteq JobPosition, Employee \sqsubseteq \neg JobPosition\}$

To sum up, we can see two points from this experiment. One is that reliability of proton ontologies certainly

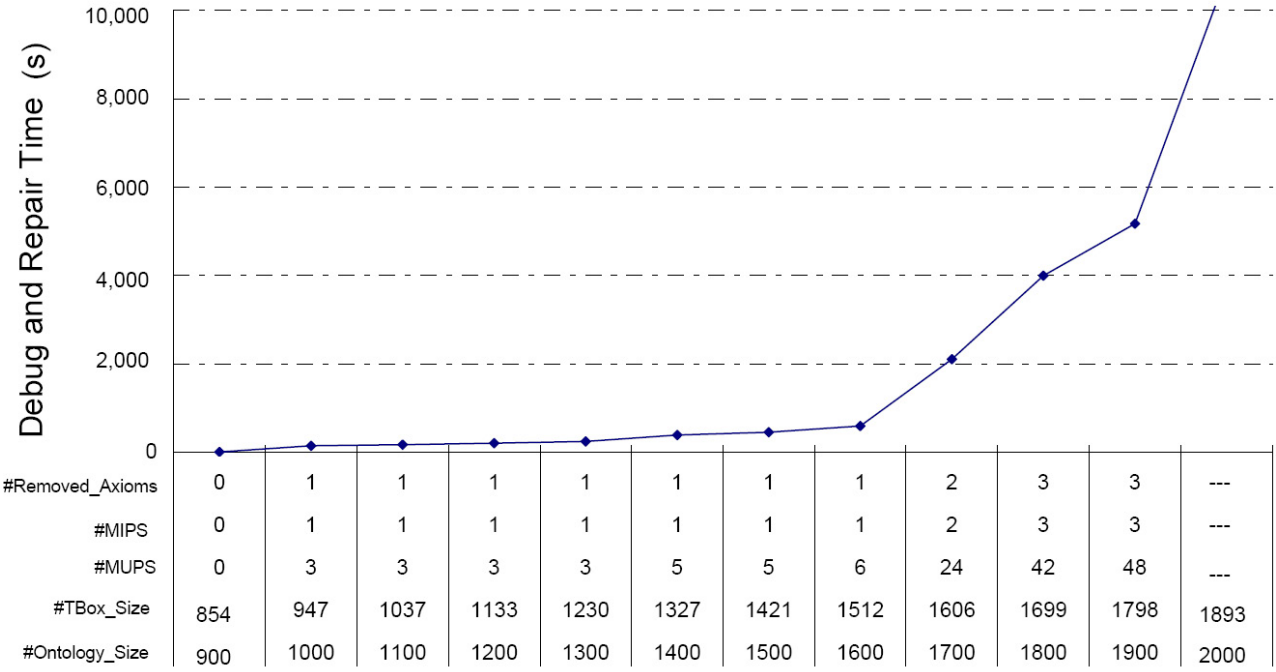
depends on the authors (experts or students) and their level of agreement (simple or complete majority) as expected in Section 5.4. Another one is that the debugging time is related to not only the TBox size but the number of unsatisfiable concepts and MUPS.

6.2 Scalability of the System

We evaluate the scalability of the system for the use case of diagnosis and repair as well as for consistent ontology evolution. Additionally, for consistent ontology evolution, we distinguish between relying on a ranking of the axioms based on scoring function and on confidence values.

6.2.1 Diagnosis and Repair

In this experiment, we evaluate the scalability by applying our algorithm on subsets of the data sets, which we incrementally increase, and observing the effect on the runtime performance in dependence on the size of the input.



#Removed_Axioms	0	1	1	1	1	1	1	1	2	3	3	---
#MIPS	0	1	1	1	1	1	1	1	2	3	3	---
#MUPS	0	3	3	3	3	5	5	6	24	42	48	---
#TBox_Size	854	947	1037	1133	1230	1327	1421	1512	1606	1699	1798	1893
#Ontology_Size	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000

Figure 6.1: Diagnosis and repair for ontology km1500_i500

For km1500_i500, the approach of this evaluation is to increase the test ontology by adding randomly 100 axioms for each iteration. While we do compute the diagnosis and repair in each iteration, in the next iteration we do not consider this repair but instead operate on the whole data set, including the previously identified contradicting axioms. Although the ontology is incoherent and inconsistent, it seems that the inconsistency is caused by incoherence from the results we can obtain in the limit resource. So we only show the results to resolve the incoherence in TBox. In Figure 6.1 we can see that the incoherence occurs when the size of ontology reaches 1,000 and the time is out after the size reaches 2,000. With the sudden increase of the number of MUPS, the debugging and repairing time goes also sharply. This is in line with what we have discussed in Section 6.1.

As for fao_glossaries, the similar approach is used comparing with km1500_i500. The only difference is that we increase the test ontology by adding a randomly chosen module for each iteration, as well as the mappings

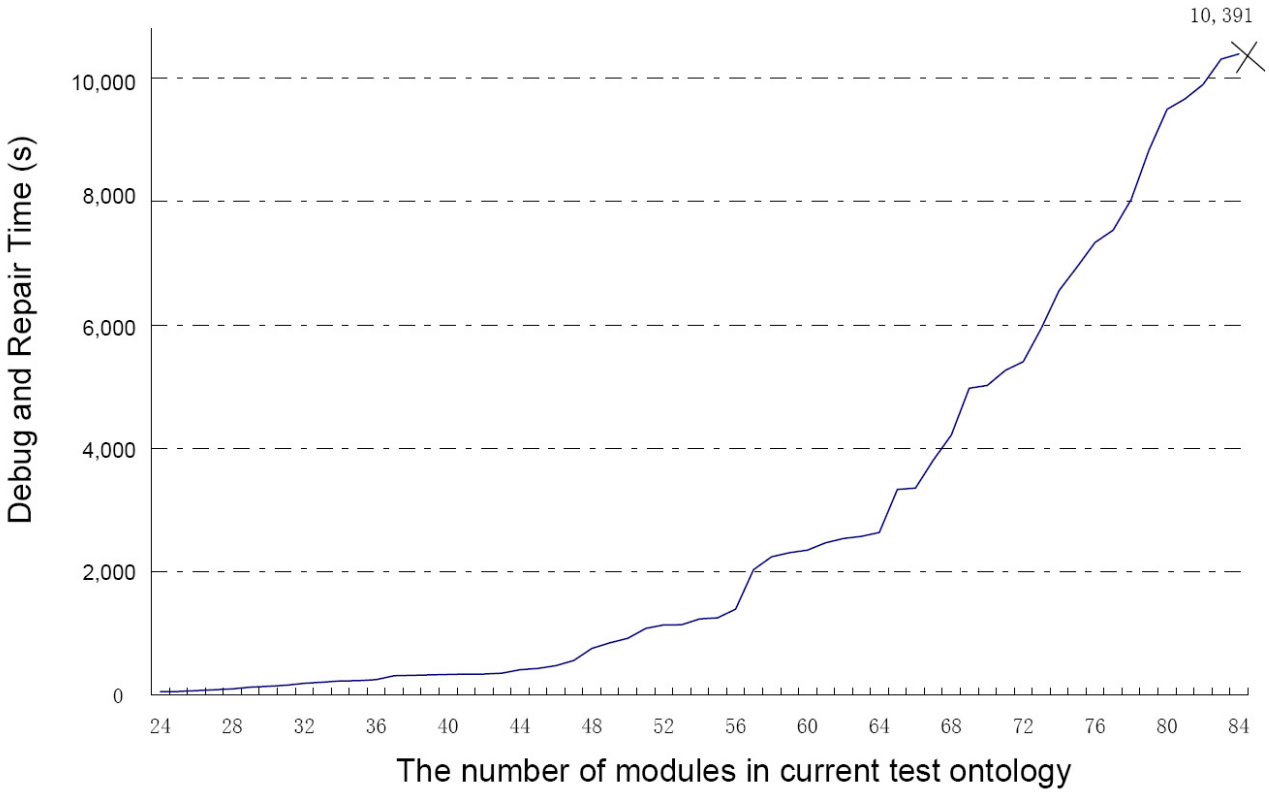


Figure 6.2: Diagnosis and repair for networked ontology (i.e. fao-glaossaries)

between each pair of chose modules. For the results shown in Figure 6.2, the incoherence begins when the number of modules reaches 24 and time is out when the 85th module is added into the test ontology. As there is not too much incoherence in this networked ontology, each test ontology with the number of modules from 24 to 84 only contains the same one unsatisfiable concept, the same one MUPS and MIPS.

From the experiment for the two ontologies we can see that the debugging time and repairing time are not only related to the TBox size but also related to the number of unsatisfiable concepts and MUPS.

6.2.2 Consistent Ontology Evolution

The setup for the experiments on consistent ontology evolution is similar to the previous one, but after each iteration we actually perform the repair to obtain a consistent ontology that serves as input for the next iteration. Thus we simulate the use case of consistent ontology evolution.

We perform one experiment where the repair is performed based on the scoring function and one based on the confidence values.

Consistent Ontology Evolution based on Scoring Function

For km1500_i500, since we add randomly the axioms, the incoherence and inconsistency begin when the size of test ontology reaches 500 which is different with the results in Section 6.2.1. We can see from the table 6.2 that time is out when the size of test ontology is 2,990 which is larger than the size of ontology can be processed by our system in Section 6.2.1.

Table 6.3 presents the results of consistent ontology evolution based on scoring function for ontology fao_glossaries. New_Added_Module shows the name of current new added module for one iteration where there is incoherence. #Modules is the number of current merged modules in the test ontology. #Mappings

Ontology_Size	TBox_size	#Unsat.	#MUPS	#MIPS	#Removed_Axioms	Ratio	Debug_Repair_Time (s)
1,200	1,145	1	1	1	1	0.50	123
2,099	2,011	8	11	2	1	0.20	2,055
2,398	2,297	8	8	2	2	0.33	1,741
2,496	2,392	1	1	1	1	0.33	611
2,595	2,487	10	14	4	1	0.14	3,726
2,694	2,582	12	12	1	1	0.25	3,178
2,893	2,768	33	33	3	3	0.30	9,262
2,990	2,861	30	30	1	1	0.25	9,074

Table 6.2: Consistent ontology evolution based on scoring function for ontology km1500_i500

New_Added_Module	#Modules	#Mappings	TBox_size	#Unsat.	#MUPS	#MIPS	#Removed_Axioms	Ratio	Debug_Repair_Time (s)
Abyssal_plain	2	0	18	1	1	1	1	0.50	1
Aquatic_ecosystem	44	802	1,434	1	2	2	1	0.14	1,062

Table 6.3: Consistent ontology evolution based on scoring function for the networked ontology

is the total number of all the mappings between the current merged modules. TBox_size here includes not only the TBox size of all current merged modules and also the corresponding mappings.

From Table 6.3 we can see that, the incoherence begins when the second module is added to the test ontology. But there is no mapping between the existing modules and this new added one (i.e. module “Abyssal_plain"). So it is obvious that the incoherence comes from the new added modules itself since different modules are independent of each other. After resolving the incoherence in the second added module, another incoherence occurs when the 44th module “Aquatic_ecosystem" is merged. After that, no new incoherence can be found until time is out when 197th module is merged. When the 196th module is added, the processed TBox size is 22,029 which includes 19,129 mappings. Thus, for the approach of consistent ontology evolution, our system can process more modules than the approach in Section 6.2.1.

The results for the two data sets show that consistent ontology evolution could process larger test ontology than the approach of diagnosis and repair. Compared with the approach of diagnosis and repair, consistent ontology evolution approach can process 1,063 more axioms for km1500_i500 and 112 more modules for networked ontology.

Consistent Ontology Evolution based on Confidence Value

In this experiment, we rely on the confidence values (if they are available for a test ontology), it is thus not necessary to get MUPS and MIPS for all the unsatisfiable concepts together to compute the scores, which is quite time-consuming. In such cases, one axiom with the lowest confidence value in one MUPS is chose to be removed. The unsatisfiable concepts can be repaired one by one.

Table 6.4 gives more details about consistent ontology evolution based on confidence values for km1500_i500 [1]. Time is out when the size of test ontology is 4,134 which is larger than the maximal size 2,990 of test ontology when applying scoring function in Section 6.2.1. While more axioms are removed to keep the coherence.

In Table 6.5, the second column is the maximal size of test ontology we can process in the limit time. The maximal size of TBox is shown in the third column. For other columns except column one, we compute the sum for each corresponding item such as the number of MUPS for all the constructed ontologies whose size is no more than the given size in column two. The second row comes from Table 6.2 which are the overall results from Algorithm 1 described in Section 4.1. For Algorithm 2, we separate the results into two parts. The first part describes the results when ontology size reaches a comparable number (i.e. 2,959) with Algorithm 1 which can process ontology with the maximal size 2,990. The second part represents the overall results of all the constructed ontologies for each test.

[1] Although there are also confidence values for this networked ontology, they are not as reliable as those for km1500_i500. We did not test the networked ontology using this ranking function (confidence value).

Ontology_Size	TBox_size	#Unsat.	#MUPS	#MIPS	#Removed_Axioms	Ratio	Debug_Repair_Time (s)
500	477	1	1	1	1	0.50	24
999	948	1	1	1	1	0.25	87
1098	1,041	2	2	2	2	0.40	136
1396	1,324	2	2	2	2	0.20	228
1694	1,603	3	4	4	4	0.26	656
1890	1,792	3	5	5	5	0.21	1,940
1985	1,885	3	3	3	3	0.27	597
2182	2,072	2	2	2	2	0.33	599
2280	2,164	1	1	1	1	0.17	510
2379	2,258	2	2	2	2	0.20	728
2477	2,347	2	4	4	2	0.14	1,156
2575	2,442	9	10	8	10	0.32	2,451
2665	2,528	6	6	6	6	0.23	1,774
2959	2,812	1	1	1	1	0.20	903
3058	2,906	1	1	1	1	0.14	964
3157	2,998	5	5	5	5	0.26	2,324
3252	3,085	1	1	1	1	0.25	1,101
3351	3,178	1	4	4	2	0.17	3,001
3449	3,272	1	2	2	2	0.29	2,030
3547	3,367	1	3	3	1	0.08	3,002
3846	3,651	5	8	8	7	0.23	5,112
3939	3,739	2	4	4	2	0.17	4,339
4037	3,836	2	4	4	3	0.27	4,054
4134	3,930	2	4	4	2	0.17	3,670

Table 6.4: Consistent ontology evolution based on confidence value for ontology km1500_i500

Algorithm	Ontology_size Max	TBox_size Max	#Unsat. Sum	#MUPS Sum	#Removed_Axioms Sum	Debug_Repair_Time Sum (s)
Algorithm 1	2,990	2,861	103	110	11	29,770
Algorithm 2	2,959	2,812	38	44	42	11,790
	4,134	3,930	59	80	68	41,388

Table 6.5: The comparison of the results based on two ranking function for ontology km1500_i500

Obviously, comparing with Algorithm 1, Algorithm 2 is more efficient by saving about 60% time to resolve incoherence when the similar maximal ontology size is reached. In the second part of Algorithm 2, it proves that its scalability (i.e., 3,930 axioms in TBox) is larger than that of Algorithm 1 (i.e., 2,861 axioms). However, the disadvantage is that more axioms have to be removed (11 for Algorithm 1 and 68 for Algorithm 2).

6.3 The analysis of meaningfulness

To measure the meaningfulness, given the removed axioms and corresponding MIPSs or MUPSs where each axiom is associated with score or confidence value, three users are asked to check whether the removed axioms make sense or not. If the majority of them agree to the same decision, we adopt this decision as the final results of meaningfulness.

More details can be seen from Table 6.6 for km1500_i500 and Table 6.7 for networked ontology, where only the distinct MIPSs and their corresponding axioms are presented to the users for checking meaningfulness. For example, in the results for km1500_i500 by applying the approach using scoring function, there are 11 distinct MIPSs for all the incoherent test ontologies. So the users need to check the 11 removed axioms.

Approach	#Removed_Axioms	#Removed_Axioms_MakeSense	#Removed_Axioms_Unknown	#Removed_Axioms_NotMakeSense	Meaningfulness
Consistent ontology evolution based on scoring function	11	9	0	2	0.82
Consistent ontology evolution based on confidence value	68	53	5	10	0.78

Table 6.6: The results of meaningfulness checking for ontology km1500_i500

For the approach of consistent ontology evolution based on confidence value, it has quite similar meaningfulness (i.e., 0.78) as the approach using scoring function (i.e., 0.82), although more axioms are removed to

keep the coherence. Besides, according to Table 6.5, we can see that, comparing with the first approach, the second one is more efficient when dealing with the similar size of TBox and it can process larger sizes of TBox within the limited resources without influencing too much of the meaningfulness.

Approach	#Removed_Axioms	#Removed_Axioms_MakeSense	#Removed_Axioms_Unknown	#Removed_Axioms_NotMakeSense	Meaningfulness
Diagnosis and Repair	1	1	0	0	1.00
Consistent ontology evolution based scoring function	2	2	0	0	1.00

Table 6.7: The results of meaningfulness checking for networked ontology

For the networked ontology, there is only 1 MIPS for all incoherent test ontologies before time is out for the first approach. There are 2 MIPSs for the second approach. It means that there is not too much contradictory information in the networked ontology. Our system works quite well for this ontology with the promising meaningfulness.

For better understanding of our results, two examples are described to show how the users make a decision of whether the removed axioms do make sense or not. The first example comes from km1500_i500 when applying the approach of consistent ontology evolution based on confidence value. One MUPS consists of the following three axioms:

(1) $model\_c \sqsubseteq \neg tool\_c$ with certainty degree 0.1

(2) $uncertainty\_model\_c \sqsubseteq \neg model\_c$ with certainty degree 1.0

(3) $model\_c \sqsubseteq tool\_c$ with certainty degree 0.13

Our algorithm selects axiom (1) to delete because its certainty degree is the smallest. However, the user thinks that this does not make sense because it is easier to accept that $model\_c$ and $tool\_c$ is disjoint than that $model\_c$ is a sub-concept of $tool\_c$.

The second example is derived from the network ontology. The axiom $target\_and\_non\_target \equiv target \sqcap non\_target$ is removed from the following MIPS:

(1) $target\_and\_non\_target \equiv target \sqcap non\_target$

(2) $non\_target \equiv \neg target$

The users agree this removed one makes sense for two reasons. One is that this removed axiom itself has not too much meaning which is contradictory according to the natural language. Another reason is that there is no doubt axiom (2) is correct.

In a word, the meaningfulness checking proves our system works well for resolving incoherence and inconsistency. It is noted that the approach of consistent ontology evolution based on confidence value can process larger test ontologies than the approach of consistent ontology evolution and diagnosis and repair. At the same time, the meaningfulness is almost as good as that of the second approach.

Chapter 7

Conclusion

7.1 Summary

In this deliverable, we have evaluated our general framework for resolving inconsistency and incoherence introduced in NeOn deliverable D1.2.1. We first provided two specific approaches for resolving incoherence and one specific approach for resolving inconsistency. We gave two algorithms to repair an incoherent ontology. The first algorithm is based on a score ordering and the second one is based on certainty degrees attached to terminology axioms. We also gave an algorithm to find one cardinality-maximal consistent subset of the ABox w.r.t. the TBox and another ontology.

The evaluations were performed against various data sets from NeOn case studies, which exhibit different forms of networking relationships. Our evaluation was done by considering the following evaluation measures: the first measure is the characterization of inconsistency, the second measure is the runtime of the algorithm and the third measure is the correctness of the results of our approach.

According to the measures above, we tested our system with some learned ontologies. Each experiment below corresponds to one measure above :

1. The first experiment is to debug some ontologies in a subset of our data sets. It can be seen that reliability of proton ontologies certainly depends on the authors (experts or students) and their level of agreement (simple or complete majority). Moreover, the size of TBox and the number of unsatisfiable concepts are two main factors that influence the performance of debugging algorithm.

2. The second experiment to check the scalability of our system, which shows that the approach of consistent ontology evolution performs better than the approach of diagnosis and repair for the similar task. Besides, the former approach can process larger test ontologies than the latter. For example, for networked ontology, 112 more modules can be processed by the former approach.

3. The last experiment is to check the meaningfulness of the removed axioms. It shows that our system can obtain quite good meaningful results. When applying confidence value as ranking function, larger TBoxes can be processed and the meaningfulness (e.g. 0.78 for km1500_i500) is almost as good as that (e.g. 0.82 for km1500_i500) of the approach using scoring function at the same time.

An important observation in the analysis of types of inconsistencies is that we are mainly confronted with incoherence on the TBox level, while ABox inconsistencies are less frequent. While one may argue that this is due to the selection of our data sets, this observation seems to be in line with other experiments performed on dealing with inconsistencies.

This leads us to the conclusion that our approach will work in practical scenarios, where TBox sizes are typically smaller than associated ABoxes.

7.2 Roadmap

The results have shown that in principle our algorithms can be applied for real-life use cases on real-life data. In the future, we intend to deploy our RaDON component in the NeOn case study prototypes, in particular integrated into the ontology learning process of the FAO case study, in which inconsistencies are most likely to occur. The integration of RaDON as an engineering component into the NeOn toolkit is already ongoing.

On the theoretical side, we intend to extend our approaches to be able to deal with other kind of networking relationships (e.g. by paying additional attention to mappings and modules) and to consider alternative notions of (in)consistency.

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.

[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.

[FFKI06] Marcelo A. Falappa, Eduardo L. Fermé, and Gabriele Kern-Isberner. On the logic of theory change: Relations between incision and selection functions. In *Proc. of ECAI'06*, pages 402–406, 2006.

[FHP+06] Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, negations and changes in ontologies. In *Proc. of AAAI'06*, pages 1295–1300, 2006.

[FPA05] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On applying the AGM theory to DLs and OWL. In *Proc. of 4th International Conference on Semantic Web (ISWC'05)*, pages 216–231. 2005.

[FS05] Gerhard Friedrich and Kostyantyn M. Shchekotykhin. A general diagnosis method for ontologies. In *Proc. of 4th International Conference on Semantic Web (ISWC'05)*, pages 232–246, 2005.

[Gin86] Matthew L. Ginsberg. Counterfactuals. *Artif. Intell.*, 30(1):35–79, 1986.

[Han94] Sven Ove Hansson. Kernel contraction. *J. Symb. Log.*, 59(3):845–859, 1994.

[HM01] Volker Haarslev and Ralf Möller. RACER system description. In *IJCAR'01*, pages 701–706, 2001.

[Hor98] Ian Horrocks. The fact system. In *Proc. of International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, pages 307–312. Springer, 1998.

[HvHH+05] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure:. A framework for handling inconsistency in changing ontologies. In *Proc. of 4th International Semantic Web Conference (ISWC'05)*, pages 353–367. Springer, 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.

[KPGS06] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca Grau, and Evren Sirin. Justifications for Entailments in Expressive Description Logics. Technical report, 2006.

[LPSV06] Joey Lam, Jeff Z. Pan, Derek Seeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In *Proc. of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06)*, pages 428 – 434, 2006.

[MLB05] Thomas Meyer, Kevin Lee, and Richard Booth. Knowledge integration for description logics. In *Proc. of 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 645–650. AAAI Press, 2005.

[MLBP06] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic alc. In *Proc. of AAAI'06*, pages 269–274, 2006.

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

[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.

[QLB06a] Guilin Qi, Weiru Liu, and David A. Bell. Knowledge base revision in description logics. In *Proc. of JELIA'06*, pages 386–398. Springer Verlag, 2006.

[QLB06b] Guilin Qi, Weiru Liu, and David A. Bell. A revision-based algorithm for handling inconsistency in description logics. In *Proc. of NMR'06*, pages 124–132, 2006.

[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 *IJCAI'03*, pages 355–362, 2003.

[Sch05] Stefan Schlobach. Diagnosing terminologies. In *Proc. of AAAI'05*, pages 670–675, 2005.

[Sha07] Zenghuan Shan. Black-box based debugging approaches for incoherent ontologies. Master's thesis, University of Karlsruhe, 2007.

[SHC06] Stefan Schlobach, Zhisheng Huang, and Ronald Cornet. Inconsistent ontology diagnosis: Evaluation. Technical report, Department of Artificial Intelligence, Vrije University Amsterdam; SEKT Deliverable D3.6.2, 2006.