Inconsistent Ontology Diagnosis and Repair

Stefan Schlobach and Zhisheng Huang
(Vrije Universiteit Amsterdam)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D3.6.3(WP3.6)
In this document, we overview the framework for inconsistent ontology diagnosis and repair, which is based on a number of new non-standard reasoning services to explain inconsistency through pinpointing. In this document, we also present a prototype of DION2, an extended system of DION (Debugger of Inconsistent ONtologies). DION is developed based a bottom-up approach to calculate pinpoints by the support of an external DL reasoner. In this document, we also discuss the implementation issues of DION2, and report preliminary experiments of DION2 with SEKT ontology learning data and SEKT legal ontologies for the inconsistent ontology debugging and repair.
Keyword list: ontology management, inconsistency diagnosis, ontology reasoning

Document Id.	SEKT/2006/D3.6.3/v1.0.0
Project	SEKT EU-IST-2003-506826
Date	Feb 9, 2007
Distribution	public

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid
Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EAD, Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

In this document, we give an overview of the framework for inconsistent ontology diagnosis and repair, which is based on a number of new non-standard reasoning services to explain inconsistency through pinpointing. DION has been developed as a bottom-up approach to calculate pinpoints by the support of an external DL reasoner.

We have implemented the prototype of DION2, an extended system of DION (Debugger of Inconsistent ONtologies). The new functionalities of DION2 are: multiple platform support, integration with KAON2, and preprocessing of inconsistent ontologies for more fine-grained debugging.

In this document, we describe two preliminary experiments in which we applied DION2 to SEKT ontology learning data and SEKT legal ontologies for their inconsistent ontology debugging and repair.

Contents

Chapter 1

Introduction

Ontologies play a crucial role in the Semantic Web (SW), as they allow the sharing of information in a semantically unambiguous way, and to reuse domain knowledge (possibly created by external sources). However, this makes SW technology highly dependent on the quality and correctness of these ontologies. Two general strategies for quality assurance are predominant, one based on developing more and more sophisticated ontology modeling tools, the second one based on logical reasoning. In this document we will focus on the latter. With the advent of expressive ontology languages such as OWL and its close relation to Description Logics (DL), state-of-the art DL reasoners can efficiently detect inconsistencies even in very large ontologies. The practical problem remains what to do in case an ontology has been detected to be locally incorrect.

There are two main ways to deal with inconsistent ontologies. One is to simply "live with" the inconsistency and to apply a non-standard reasoning method to obtain meaningful answers in the presence of inconsistencies. Such an approach is taken in [11]. An alternative approach is to resolve (or: "debug") the error whenever an inconsistency is encountered. In this document we focus on this *debugging* process, and we will focus on the terminological part of ontologies (and talk about debugging of terminologies). We will introduce the formal foundations for debugging and diagnosis of logically incorrect terminologies, more precisely the notions of *minimal unsatisfiability-preserving sub-TBoxes* (abbreviated MUPS) and *minimal incoherence-preserving sub-TBoxes* (MIPS) as the smallest subsets of axioms of an incoherent terminology preserving unsatisfiability of a particular, respectively of at least one unsatisfiable concept.

Our approach to diagnosing incoherent terminologies is based on traditional *model-based diagnosis* which has been studied over many years in the AI community [20]. Here the aim is to find minimal fixes, i.e. in our case minimal subsets of an terminology that need to be repaired or removed to render a terminology logically correct, and therefore usable again. We will see that in Reiter's terminology, MIPS and MUPS correspond to minimal conflict sets.

In SEKT deliverable D3.6.1 [26], we describe two algorithms for debugging, a

bottom-up method using the support of an external reasoner, and an implementation of a specialized top-down algorithm. The former is based on the systematic enumerations of terminologies of increasing size based on selection functions on axioms, the latter is based on Boolean minimization of labels in a labeled tableau calculus. Both methods have been implemented as prototypes. The prototype for the informed bottom-up approach is called DION (Debugger of Inconsistent ONtologies)[1], the prototype of the specialized top-down method is called MUPSTER.

In [25], we provide a detailed evaluation of our methods. We perform a set of *controlled benchmark experiments* to get a better understanding of the computational properties of the debugging problem and our algorithms for solving it. The combined results of the case-study and the controlled experiments show that on the one hand, debugging is useful in practice, but that on the other hand we cannot guarantee that our tools will always find explanations in a reasonable time. The most important criteria will turn out to be the size and complexity of the definitions and the number of modeling errors.

In this document, we will report the approach of DION only, because DION more flexible, expressive and easy to adapt [25]. Therefore, the extension of MUPSter is discontinued.

This document is organized as follows: Chapter 2 proposes a framework of inconsistent ontology diagnosis and repair. Chapter 3 describes the prototypes of the implementation issues of DION2. Chapter 4 reports the experiments of DION2 with SEKT ontology learning data. Chapter 5 discusses further work and concludes the document.

[1] http://wasp.cs.vu.nl/sekt/dion

Chapter 2

Debugging Incoherent Terminologies

Description Logics are a family of well-studied set-description languages which have been in use to formalize knowledge for over two decades. They have a well-defined model theoretic semantics, which allows for the automation of a number of reasoning services.

2.1 Logical Errors in Description Logic Terminologies

For a detailed introduction to Description Logics we point to the second chapter of the DL handbook [2]. Briefly, in DL concepts will be interpreted as subsets of a domain, and roles as binary relations. Let, throughout the paper, $\mathcal{T} = \{ax_1, \ldots, ax_n\}$ 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. We will also use terminological axioms of the form $C = D$ and disjointness statements *disjoint(C,D)* between two concepts C and D, which are simple abbreviations of $(C \sqsubseteq D)\&(D \sqsubseteq C)$, and $C \sqsubseteq \neg D$ respectively.[1] Most DL systems also allow for assertional axioms in a so-called ABox. In this paper, ABoxes will not be considered. Throughout the paper the term *ontologies* will refer to general knowledge bases which possibly include both terminological and assertional knowledge. The term *terminology* is solely used in the technical sense of a DL TBox.

2.1.1 Unsatisfiability and Incoherence

Let $\mathcal{U}$ be a finite set of objects, called the universe. A mapping $\mathcal{I}$, which interprets DL concepts as subsets of $\mathcal{U}$ is a *model* of a terminological axiom $C \sqsubseteq D$, if, and only if, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A *model for a TBox* $\mathcal{T}$ is an interpretation which is a model for all axioms in $\mathcal{T}$. Based on these semantics a TBox can be checked for *incoherence*, i.e., whether there

[1] The latter requires negation in the language, and a careful treatment of unfolding (which might either become incomplete or non-terminating).

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

Table 2.1: A small (incoherent) TBox $\mathcal{T}_1$

are *unsatisfiable* concepts: concepts which are necessarily interpreted as the empty set in all models of the TBox. More formally

1. A concept C is *unsatisfiable* w.r.t. a terminology $\mathcal{T}$ if, and only if, $C^{\mathcal{I}} = \varnothing$ for all models $\mathcal{I}$ of $\mathcal{T}$.

2. A TBox $\mathcal{T}$ is *incoherent* if there is a concept name in $\mathcal{T}$, which is unsatisfiable.

Conceptually, these cases often point to modeling errors because we assume that a knowledge modeler would not specify something like an impossible concept in a complex way.

Table 2.1 demonstrates this principle. Consider the (incoherent) TBox $\mathcal{T}_1$, where A, B and C, as well as $A_1, \ldots, A_7$ are concept names, and r and s roles. Satisfiability testing returns a set of unsatisfiable concept names $\{A_1, A_3, A_6, A_7\}$. Although this is still of manageable size, it hides crucial information, e.g., that unsatisfiability of A_1 depends, among others, on unsatisfiability of A_3, which is in turn unsatisfiable because of the contradictions between A_4 and A_5. We will use this example later in this paper to explain our debugging methods.

2.1.2 Unfoldable $\mathcal{ALC}$ TBoxes

In this paper we study ways of *debugging and diagnosing* of incoherence and unsatisfiability in DL terminologies. The general ideas can easily be extended to inconsistency of ontologies with assertions as suggested in [25]. As the evaluation in this paper will be about terminological debugging only, we will restrict the technical definitions to the necessary notions.

Whereas the definitions of debugging were independent of the choice of a particular Description Logic, we will later present algorithms for the Description Logic $\mathcal{ALC}$, and unfoldable TBoxes, in particular.

$\mathcal{ALC}$ is a simple yet relatively expressive DL with conjunction $(C \sqcap D)$, disjunction $(C \sqcup D)$, negation $(\neg C)$ and universal $(\forall r.C)$ and existential quantification $(\exists r.C)$, where the interpretation function is extended to the different language constructs as follows:

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = U \setminus C^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{d \in U \mid \exists e \in U : (d, e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{d \in U \mid \forall e \in U : (d, e) \in R^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$$

A TBox is called *unfoldable* if the left-hand sides of the axioms (the defined concepts) are atomic and unique, and if the right-hand sides (the definitions) contain no direct or indirect reference to the defined concept [19].

2.2 Framework for debugging and diagnosis

We now introduce a theory of debugging and diagnosis and link it to description logic-based systems. In this case a diagnosis is a smallest set of axioms that needs to be removed or corrected to render a specific concept or all concepts satisfiable.

In some situations, terminologies can contain a large number of unsatisfiable concepts. This can occur for example when terminologies are the result of a merging process of separately developed terminologies, or when closure axioms (i.e. disjointness statements and universal restrictions) are added to terminologies. Unsatisfiability propagates, i.e. one unsatisfiable concept may cause many other concepts to become unsatisfiable as well. As it is often not clear to a modeler what concepts are the root cause of unsatisfiability, we also describe a number of heuristics that help to indicate reasonable starting points for debugging an terminology.

2.2.1 Model-based Diagnosis

The literature on model-based diagnosis is manifold, but we focus on the seminal work of Reiter [20], and [8], which corrects a small bug in Reiter's original algorithm. We refer the interested reader to a good overview in [5].

Reiter introduces a diagnosis of a system as the smallest set of components from that system with the following property: the assumption that each of these components is faulty (together with the assumption that all other components are behaving correctly) is consistent with the system description and the observed behavior. In other words: assuming correctness of any one of the components in a diagnosis would cause inconsistency between the system description and the observed behavior.

To apply this definition to a description logic terminology, we regard the terminology as the *system* to be diagnosed, and the axioms as the *components* of this system. If we look at the example terminology from Table 2.1, the *system description* states that it is coherent (i.e. all concepts are satisfiable), but the *observation* is that A_1, A_3, A_6, and A_7 are unsatisfiable. In Reiter's terminology, a minimal set of axioms that need to be

removed (or better fixed) is called a diagnosis. This adaptation of Reiter's method leads to the following definition of terminological diagnosis.

Definition 1 *Let $\mathcal{T}$ be an incoherent terminology. A* (terminological) diagnosis *for the incoherence problem of $\mathcal{T}$ is a minimal set of axioms $\mathcal{T}' \subseteq \mathcal{T}$ such that $\mathcal{T} \setminus \mathcal{T}'$ is coherent. Similarly, a diagnosis for unsatisfiability of a single concept A in $\mathcal{T}$ is any minimal subset $\mathcal{T}' \subseteq \mathcal{T}$, such that A is satisfiable w.r.t. $\mathcal{T} \setminus \mathcal{T}'$.*

Reiter provides a generic method to calculate diagnoses on the basis of conflict sets and their minimal hitting sets. A conflict set is a set of components that, when assumed to be fault free, lead to an inconsistency between the system description and observations. A conflict set is minimal if and only if no proper subset of it is a conflict set. The minimal conflict sets (w.r.t. coherence) for the system in Table 2.1 are $\{ax_1, ax_2\}$, $\{ax_3, ax_4, ax_5\}$, and $\{ax_4, ax_7\}$.

A hitting set H for a collection of sets C is a set that contains at least one element of each of the sets in C. Formally: $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \emptyset$ for each $S \in C$. A hitting set is minimal if and only if no proper subset of it is a hitting set. Given the conflict sets above, the minimal hitting sets are: $\{ax_1, ax_3, ax_7\}$, $\{ax_1, ax_4\}$, $\{ax_1, ax_5, ax_7\}$, $\{ax_2, ax_3, ax_7\}$, $\{ax_2, ax_4\}$, and $\{ax_2, ax_5, ax_7\}$.

Reiter shows that the set of diagnoses actually corresponds to the collection of minimal hitting sets for the minimal conflict sets. Hence, the minimal hitting sets given above determine the diagnoses for the system w.r.t. coherence.

2.2.2 Debugging

As previously mentioned, the theory of diagnosis is built on minimal conflict sets. But in the application of diagnosis of erroneous terminologies, these minimal conflict sets play a role of their own, as they are the prime tools for debugging, i.e. for the identification of potential errors. For different kind of logical contradictions we introduce several different notions based on conflict sets, the MUPS for unsatisfiability of a concept, the MIPS for incoherence of a terminology.

Minimal unsatisfiability-preserving sub-TBoxes (MUPS)
In [22] we introduced the notion of Minimal Unsatisfiability Preserving Sub-TBoxes (MUPS) to denote minimal conflict sets. Unsatisfiability-preserving sub-TBoxes of a TBox $\mathcal{T}$ and an unsatisfiable concept A are subsets of $\mathcal{T}$ in which A is unsatisfiable. In general there are several of these sub-TBoxes and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

Definition 2 *A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a minimal unsatisfiability preserving sub-TBox (MUPS) for A in $\mathcal{T}$ if A is unsatisfiable in $\mathcal{T}'$, and A is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$.*

We will abbreviate the set of MUPS of $\mathcal{T}$ and A by $mups(\mathcal{T}, \text{A})$. MUPS for our example TBox $\mathcal{T}_1$ and its unsatisfiable concepts are:

$mups(\mathcal{T}_1, A_1)$: $\{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$

$mups(\mathcal{T}_1, A_3)$: $\{\{ax_3, ax_4, ax_5\}\}$

$mups(\mathcal{T}_1, A_6)$: $\{\{ax_1, ax_2, ax_4, ax_6\}, \{ax_1, ax_3, ax_4, ax_5, ax_6\}\}$

$mups(\mathcal{T}_1, A_7)$: $\{\{ax_4, ax_7\}\}$

In the terminology of Reiter's diagnosis each $mups(\mathcal{T}, A)$ is a collection of minimal conflict sets w.r.t. satisfiability of concept A in TBox $\mathcal{T}$.

Remember that a diagnosis is a minimal hitting set for a collection of conflict sets. Hence, from the MUPS, we can also calculate the diagnoses for unsatisfiability of concept A in TBox $\mathcal{T}$, which we will denote $\Delta_{\mathcal{T},A}$.

$\Delta_{\mathcal{T}_1,A1} : \{\{ax_1\}, \{ax_2, ax_3\}, \{ax_2, ax_4\}, \{ax_2, ax_5\}\}$

$\Delta_{\mathcal{T}_1,A3} : \{\{ax_3\}, \{ax_4\}, \{ax_5\}\}$

$\Delta_{\mathcal{T}_1,A6} : \{\{ax_1\}, \{ax_4\}, \{ax_6\}, \{ax_2, ax_3\}, \{ax_2, ax_5\}\}$

$\Delta_{\mathcal{T}_1,A7} : \{\{ax_4\}, \{ax_7\}\}$

Minimal incoherence-preserving sub-TBoxes (MIPS)
MUPS are useful for relating sets of axioms to the unsatisfiability of specific concepts, but they can also be used to calculate MIPS, which relate sets of axioms to the incoherence of a TBox in general (i.e. unsatisfiability of at least one concept in a TBox).

Definition 3 *A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a minimal incoherence preserving sub-TBox (MIPS) for A in $\mathcal{T}$ if $\mathcal{T}'$ is incoherent, every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent.*

This means that MIPS are minimal subsets of an incoherent TBox preserving unsatisfiability of at least one atomic concept. The set of MIPS for a TBox $\mathcal{T}$ is abbreviated with $mips(\mathcal{T})$. For $\mathcal{T}_1$ we get 3 MIPS: $mips(\mathcal{T}_1) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$

Analogous to MUPS, each element of $mips(\mathcal{T})$ is a minimal conflict set w.r.t. incoherence of TBox $\mathcal{T}$. Hence, from $mips(\mathcal{T})$, a diagnosis for coherence of $\mathcal{T}$ can be calculated, which we denote as $\Delta_{\mathcal{T}}$. From these definitions, we can determine the diagnosis for coherence of $\mathcal{T}_1$:

$\Delta_{\mathcal{T}1} = \{\{ax_1, ax_4\}, \{ax_2, ax_4\}, \{ax_1, ax_3, ax_7\}, \{ax_2, ax_3, ax_7\}, \{ax_1, ax_5, ax_7\}, \{ax_2, ax_5, ax_7\}\}$

The number of MUPS a MIPS is a subset of determines the number of unsatisfiable concepts of which it might be the cause. We will call this number the *MIPS-weight*.

In the example terminology $\mathcal{T}_1$ we found six MUPS and three MIPS. The MIPS $\{ax_1, ax_2\}$ is equivalent to one of the MUPS for A_1, $\{ax_1, ax_2\}$, and a proper subset of a MUPS

for A_6, $\{ax_1, ax_2, ax_4, ax_6\}$. Hence, the weight of MIPS $\{ax_1, ax_2\}$ is two. In the same way we can calculate the weights for the other MIPS: the weight of $\{ax_3, ax_4, ax_5\}$ is three, the weight of $\{ax_4, ax_7\}$ is one. Intuitively, this suggests that the combination of the axioms $\{ax_3, ax_4, ax_5\}$ is more relevant than $\{ax_4, ax_7\}$.

Weights are easily calculated, and play an important role in practice to determine relative importance within the set of MIPS, as we experienced in several case studies.

Pinpoints

Experiments described in [21] indicated that calculating diagnoses from MIPS and MUPS is simple, but computationally expensive, and often impractical for real-world terminologies. For this purpose, we introduced in [23] the notion of a *pinpoint* of an incoherent terminology $\mathcal{T}$, in order to approximate the set of diagnoses. The definition of the set of pinpoints is a procedural one, following a heuristic to insure that *most* pinpoints will indeed be diagnoses. However, there is no guarantee of minimality, so that not every pinpoint is not necessarily diagnosis.

To define pinpoints we need the notion of a core: MIPS-weights provide an intuition of which combinations of axioms lead to unsatisfiability. Alternatively, one can focus on the occurrence of the individual axioms in MIPS, in order to predict the likelihood that an individual axiom is erroneous. We define cores as sets of axioms occurring in several MIPS. The more MIPS such a core belongs to, the more likely its axioms will be the cause of contradictions.

Definition 4 *A non-empty subset of the intersection of n different MIPS in $mips(\mathcal{T})$ (with $n \geq 1$) is called a MIPS-core of arity n (or simply n-ary core) for $\mathcal{T}$.*

For our example TBox $\mathcal{T}_1$ we find one 2-ary core, $\{ax_4\}$ of size 1. The other axioms in the MIPS are 1-ary cores. Pinpoints are defined in a structural way.

Definition 5 *Let $mips(\mathcal{T})$ be the set of MIPS of $\mathcal{T}$, i.e. a collection of sets of axioms. The set of possible outputs of the following procedure will be called the* set of pinpoints.

Let $M := mips(\mathcal{T})$ be the collection of MIPS for $\mathcal{T}$, $P = \varnothing$:
(1) Choose in M an arbitrary core $\{ax\}$ of size 1 with maximal arity.
(2) Then, remove from M any MIPS containing $\{ax\}$
(3) $P := P \cup \{ax\}$

Repeat steps (1) to (3) until $M = \varnothing$. The set P is then called a pinpoint *of the terminology.*

As step (1) contains a non-deterministic choice, there is no unique *pinpoint* but a set possible of possible outputs of the algorithm: the set of pinpoints.

For our example TBox $\mathcal{T}_1$ with $mips(\mathcal{T}_1)$ = $\{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$ we first take the 2-ary core, $\{ax_4\}$. Removing the MIPS containing ax_4 leaves $\{ax_1, ax_2\}$. Hence, there is a non-deterministic choice: if we choose ax_1 to continue $\{ax_4, ax_1\}$ is the calculated pinpoint, otherwise $\{ax_4, ax_2\}$. Both are diagnoses of $\mathcal{T}_1$.

Chapter 3

DION2: a Bottom-Up Approach of Ontology Diagnosis

3.1 General Idea

In this chapter we describe an informed bottom-up algorithm to calculate MUPS with the support of an external DL reasoner. The main advantage of this approach is that it can deal with any DL-based ontology supported by an external reasoner. Currently there exist several well-known DL reasoners, like RACER[1], FaCT++[2], and Pellet[3], each of which has been proved to be reliable and stable. They support various DL-based ontology languages, including OWL-DL.

Given an unsatisfiable concept A and a terminology $\mathcal{T}$ MUPS can be systematically calculated by checking whether A is unsatisfiable in subsets $\mathcal{T}'$ of $\mathcal{T}$ of increasing size. Such a procedure is easy to implement, but infeasible in practice. Even very simple real-world terminology we considered in [25] have an average size of 5 axioms per MUPS and 417 axioms, which requires over 10^{11} calls to the external reasoner.

This observation implies that one has to control the subsets of $\mathcal{T}$ that are checked for satisfiability of A by means of a *selection function*. Such a selection function selects increasingly large subsets which are heuristically chosen to be relevant additions to the currently selected subset. Although this approach is not guaranteed to give us the complete solution set of MUPS it provides an efficient approach for debugging inconsistent terminologies. We will now formally introduce the core notions of selection functions and relevance.

Given a terminology $\mathcal{T}$ and an axiom ax, a *selection function* s is a function which returns a linearly ordered collection of subsets of $\mathcal{T}$. More formally, for an ontology

[1] http://www.sts.tu-harburg.de/∼r.f.moeller/racer/
[2] http://owl.man.ac.uk/factplusplus/
[3] http://www.mindswap.org/2003/pellet/

language $\mathbf{L}$, a selection function s is a mapping $s : \mathcal{P}(\mathbf{L}) \times \mathbf{L} \times \mathbb{N} \rightarrow \mathcal{P}(\mathbf{L})$ such that $s(\mathcal{T}, \phi, k) \subseteq \mathcal{T}$.

In [12] we defined two different selection functions, the most simple one based on co-occurrence of concept names in axioms. As in this document we focus on unfoldable TBoxes,[4] we introduce a slightly more complex selection function here. The basic idea is that an axiom ax is relevant to a concept name A if, and only if, A occurs on the left-hand side of ax. In a way this variant of the bottom-up approach mimics the unfolding procedure in order to restrict the number of tests needed. This is the one implemented in the DION system.

We use $\mathcal{V}_c(ax)$ ($\mathcal{V}_c(C)$) to denote the set of concept names that appear in an axiom ax (in a concept C, respectively). Concept-relevance is defined as follows:

Definition 6 *An axiom ax is* concept-relevant to a concept or an axiom ϕ *iff*
(i) $\mathcal{V}_c(C_1) \cap \mathcal{V}_c(\phi) \neq \varnothing$ if the axiom ax has the form $C_1 \sqsubseteq C_2$,
(ii) $\mathcal{V}_c(C_1) \cap \mathcal{V}_c(\phi) \neq \varnothing$ or $\mathcal{V}_c(C_2) \cap \mathcal{V}_c(\phi) \neq \varnothing$ if the axiom ax has the form $C_1 = C_2$,
(iii) $\mathcal{V}_c(C_1) \cap \mathcal{V}_c(\phi) \neq \varnothing$ or $V_c(C_2) \cap V_c(\phi) \neq \varnothing$ if the axiom ax has the form $disjoint(C_1, C_2)$.

Note that this approach is a syntactic one because, for example, the axiom $\neg D \sqsubseteq \neg C$ is treated differently from the axiom $C \sqsubseteq D$.

Based on this particular relevance function we can now, for a terminology $\mathcal{T}$ and a concept A, define a selection function s as follows:

Definition 7 *The* concept-relevance based selection function *for a TBox $\mathcal{T}$ and a concept A is defined as*

(i) $s(\mathcal{T}, A, 0) = \varnothing$*;*
(ii) $s(\mathcal{T}, A, 1) = \{ax \mid ax \in \mathcal{T}$ *and ax is concept-relevant to* $A\}$*;*
(iii) $s(\mathcal{T}, A, k) = \{ax \mid ax \in \mathcal{T}$ *and ax is concept-relevant to an element in* $s(\mathcal{T}, A, k-1)\}$ *for* $k > 1$.

3.2 Algorithms

We use an informed bottom-up approach to obtain MUPS. In logics and computer science, an increment-reduction strategy is often used to find minimal inconsistent sets [6]. Under this approach, the algorithm first finds a collection of inconsistent subsets of an inconsistent set, before it removes redundant axioms from this subsets. Similarly, a heuristic procedure for finding MUPS of a TBox $\mathcal{T}$ and an unsatisfiable concept-name A consists of the following three stages:

[4] Remember that the top-down is defined for unfoldable TBoxes only.

$k := 0$
$M(\mathcal{T}, A) := \varnothing$
repeat
  $k := k + 1$
until A unsatisfiable in $s(\mathcal{T}, A, k)$ (*)
$\Sigma := s(\mathcal{T}, A, k) - s(\mathcal{T}, A, k-1)$
$S := s(\mathcal{T}, A, k-1)$
$W := \{S\}$
for all $\phi \in \Sigma$ **do**
  for all $S' \in W$ **do**
    if A satisfiable in $S' \cup \{\phi\}$ and $S' \cup \{\phi\} \notin W$ **then**
      $W := W \cup \{S' \cup \{\phi\}\}$
    end if
    if A unsatisfiable in $S' \cup \{\phi\}$ and $S' \cup \{\phi\} \notin M(\mathcal{T}, A)$ **then**
      $M(\mathcal{T}, A) := M(\mathcal{T}, A) \cup \{S' \cup \{\phi\}\}$
    end if
  end for
end for
$M(\mathcal{T}, A) := \mathbf{MinimalityChecking}(mups(\mathcal{T}, A))$
return $M(\mathcal{T}, A)$

Figure 3.1: **MUPS_bottomup(**$\mathcal{T}, A$)

- **Expansion**: Use a relevance-based selection function to find two subsets Σ and S of $\mathcal{T}$ such that a concept A is satisfiable in S and unsatisfiable in $S \cup \Sigma$.
- **Increment**: Enumerate subsets of Σ to obtain the sets S" such that the concept A is unsatisfiable in S" $\cup S$. We call those sets A-*unsatisfiable sets*.
- **Reduction**: Remove redundant axioms from those A-unsatisfiable sets to get MUPS.

Figure 3.1 describes an algorithm **MUPS_bottomup(**$\mathcal{T}, A$) based on this strategy to calculate MUPS. The algorithm first finds two subsets Σ and S of $\mathcal{T}$ by increasing the relevance degree k on the selection function until A is unsatisfiable in $S \cup \Sigma$. Compared with $\mathcal{T}$, the set Σ can be expected to be relatively small. The algorithm then builds the power-set of Σ to get A-unsatisfiable sets by adding an axiom $\phi \in \Sigma$ each time in the loop to the sets S' in the working set W. If A is satisfiable in $S' \cup \{\phi\}$, then the set $S' \cup \{\phi\}$ is added into the working set to build up the union of each elements of the power-set of Σ with the set S.[5] If A is unsatisfiable in $S' \cup \{\phi\}$, then add the set $S' \cup \{\phi\}$ into the resulting set $M(\mathcal{T}, A)$ instead of the working set W. This avoids the calculation of the full power-set of Σ because any superset of $S' \cup \{\phi\}$ in which A is unsatisfiable is pruned.

[5]Namely $\{S'/S | S' \in W\} \subseteq \mathcal{P}(\Sigma)$

Figure 3.2: **MinimalityChecking(M($\mathcal{T}$, A))**

for all $M \in M(\mathcal{T}, A)$ **do**
  $M' := M$
  for all $ax \in M'$ **do**
    if A unsatisfiable in $M' - \{ax\}$ **then**
      $M' := M' - \{ax\}$
    end if
  end for
  $M(\mathcal{T}, A) := M(\mathcal{T}, A) - \{M\} \cup \{M'\}$
end for
return $M(\mathcal{T}, A)$

Finally, by checking minimality we obtains MUPS. The procedure to check minimality of the calculated subsets of $\mathcal{T}$ is described in Figure 3.2.

Proposition 3.2.1 *The algorithm* **MUPS_bottomup($\mathcal{T}$, A)** *of figure 3.1 is sound. This means that it always returns MUPS, i.e. that* $M(\mathcal{T}, A) \subseteq mups(\mathcal{T}, A)$*, for any output* $M(\mathcal{T}, A)$.

PROOF. It follows from the construction of the sets in the collection $M(\mathcal{T}, A)$ in **MUPS_bottomup($\mathcal{T}$, A)** that the concept A is always unsatisfiable for any element S in $M(\mathcal{T}, A)$. Otherwise it would have never been added in the first place. Minimality is enforced by the procedure **MinimalityChecking(M($\mathcal{T}$, A))**. □

Take our running example. To calculate $M(\mathcal{T}_1, A_1)$, the algorithm first gets the set $\Sigma = \{ax_2, ax_3\} = \{ax_1, ax_2, ax_3\} - \{ax_1\}$. Thus, $M(\mathcal{T}_1, A_1) = \{\{ax_1, ax_2\}\}$. What has to be noted is that the algorithm cannot find that $S_1 = \{ax_1, ax_3, ax_4, ax_5\}$ is a MUPS for $\mathcal{T}_1$ and A_1. This points to the incompleteness of our algorithm. The problem is the stopping condition of the expansion phase (denoted by $(*)$ in the algorithm). This condition means that only the MUPS with axioms with maximal relevance with regard to the unsatisfiable concept will be found. In principle, the rigid stopping condition $(*)$ in **MUPS_bottomup($\mathcal{T}$, A)** could easily be replaced by a full expansion, i.e. by a condition which requires saturation of the selection process. However, as the primary goal of our implementation was practical applicability, the **MUPS_bottomup($\mathcal{T}$, A)** algorithm is implemented in DION as described above.

3.3 Implementation of DION2

3.3.1 General Consideration

We have implemented a prototype DION as a debugger of inconsistent ontologies in [26]. The system is implemented as an intelligent interface between an application and state-of-the art description logic reasoners and provides server-side functionality in terms of an XML-based interface for uploading an inconsistent ontology and posing queries for debugging. Requests to the server are analyzed by the main control component that also transforms queries into the underlying query processing. The main control element also interacts with the ontology repository and ensures that the reasoning components are provided with the necessary information and coordinates the information flow between the reasoning components.

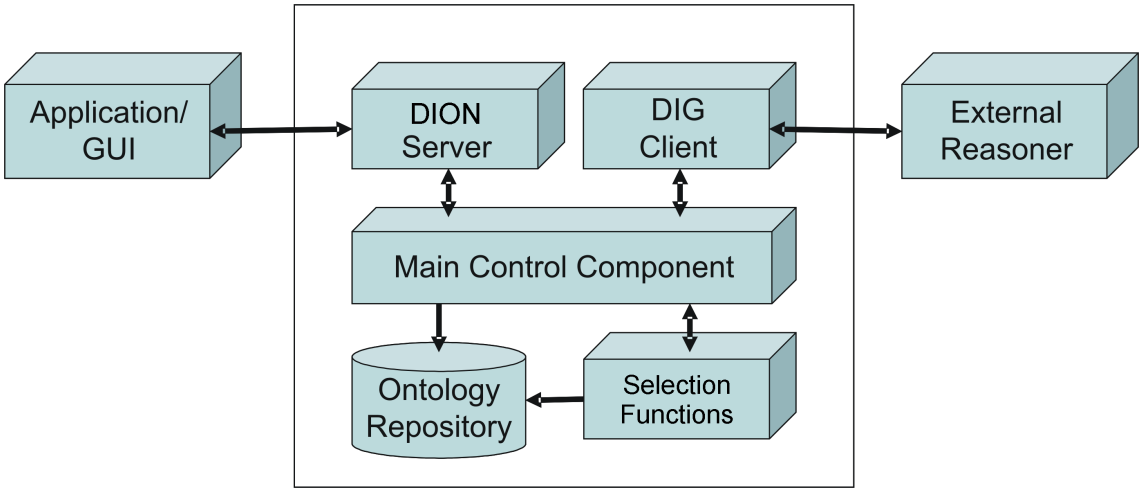

Figure 3.3: Architecture of DION.

An overview of the DION architecture is shown in Figure 3.3. It has the following components:

- **DION Server**: The DION server acts as a server which deals with requests from other ontology applications.
- **Main Control Component**: The main control component performs the main processing, like query analysis, query pre-processing, and interacting with the ontology repositories.
- **Selection Functions**: The relevance-based selction function component provides heuristic facilities to evaluate the queries.

- **DIG Client**: DION's DIG client is the standard XDIG client, which calls external DL reasoners which support the DIG interface to obtain the standard DL reasoning capabilities.
- **Ontology Repositories**: The ontology repositories are used to store ontologies and other system specifications.

The DION prototype is implemented in Prolog and uses the XDIG interface [13], an extended DIG description logic interface for Prolog[6]. DION is designed to be a simple API for a general debugger with inconsistent ontologies. It supports extended DIG requests from other ontology applications, including OWL and DIG [3][7]. This means that DION can be used as an interface of an inconsistent ontology debugger as it supports the functionality of the underlying reasoner by just passing requests on and provides reasoning functionalities if needed. Therefore, the implementation of DION will be independent of those particular applications or systems. DION2 is an extension of DION.

3.3.2 Functionalities

The prototype of DION2 has the following characteristics:

- **Debugging Support**: calculates MUPS, MIPS, cores, and pinpoints
- **Interface Support**: supports the XDIG interface, an extended DIG interface.
- **Ontology Languages**: supports the DIG format as well as the OWL language. Ontological data in the OWL format is translated automatically by the XDIG component 'owl2dig'.
- **Multiple Platforms**: supports the Windows/Linux/Unix platforms
- **Integration with external DL reasoners**: integrated with external DL reasoners, like RACER and KAON2, via their DIG interfaces.
- **Preprocessing Support**: supports more fine-grained debugging by pre-processing ontology data.

3.3.3 Installation and Test Guide

1. **Download**: The DION2 package is available from the DION website:

   `http://wasp.cs.vu.nl/sekt/dion/dion2.zip`

[6] http://wasp.cs.vu.nl/sekt/dig
[7] http://dl.kr.org/dig/

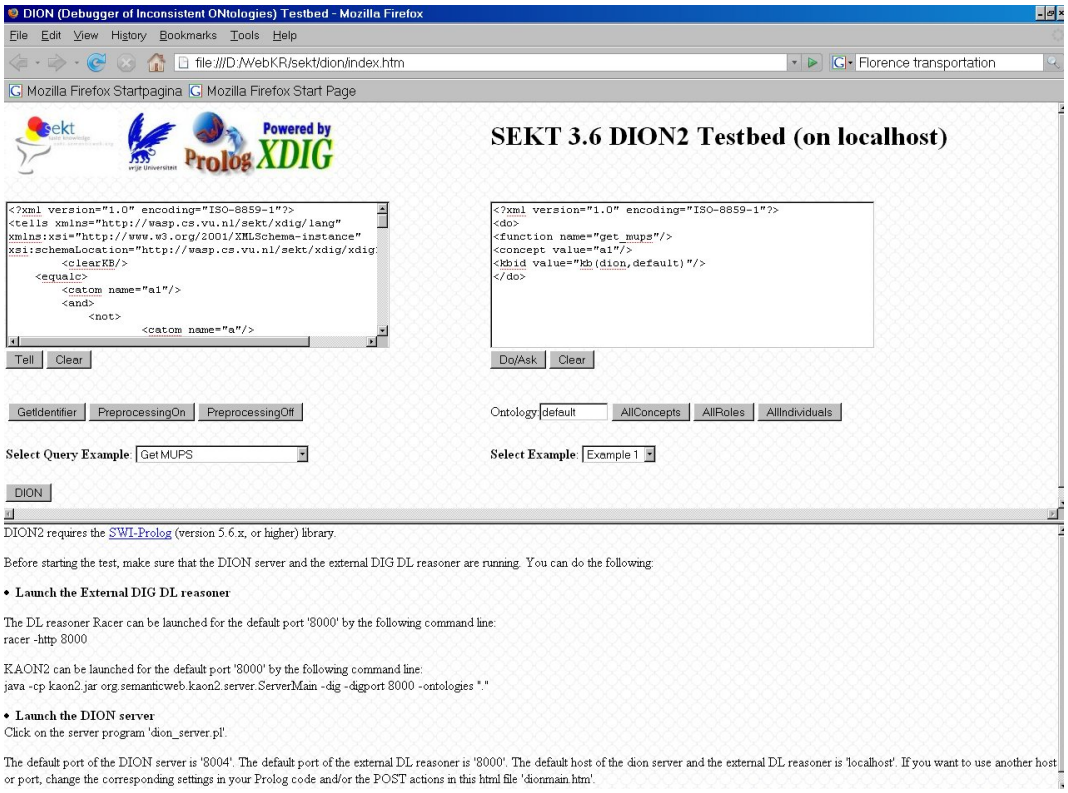

Figure 3.4: DION2 testbed

Unzip the DION package into a directory.

2. **Installation of SWI-Prolog**: PION requires that SWI-Prolog (version 5.6.0 or higher) has been installed on your computers. It can be downloaded from the SWI-Prolog website:

   `http://www.swi-prolog.org`

3. **Installation of External DL Reasoners**: DION requires an external DL reasoner which supports the DIG interface, like RACER (version 1.7.14 or higher) or KAON2. Other DL reasoners may work for DION2 if they support the DIG DL interface, however, they have not yet been tested.

3.3.4 DION2 Testbed

The DION2 testbed 'dion_test.htm' is a DION2 client with a graphical interface, which is designed as a webpage. Therefore it can be launched from a web browser which supports Javascript. A screenshot of the DION2 testbed, is shown in Figure 3.4.

The current version of the DION2 testbed supports tests for which both the DION2 server (with default port: 8004) and the external DL reasoner (with the default port: 8000) are running on the localhost. The default hostname of the DION server and the external DL reasoner is 'localhost'. For a DION2 server which runs on a remote host, change the host and port data in the 'dionmain.htm' file. Namely, replace the URL 'http://127.0.0.1:8004' with another valid URL.

Before starting the DION2 test, make sure that the DION2 server and the external DL reasoner (i.e. RACER) are running at the host with the correct ports.

1. **Launch RACER**: Racer can be launched by the following command:

   ```
   racer -http 8000
   ```

   Alternatively, click on the file 'racer8000.bat' if the DION2 downloaded package includes the reasoner RACER and the batch file. KAON2 can be launched with its DIG interface at the port '8000' by the following command:

   ```
   java -cp kaon2.jar  org.semanticweb.kaon2.server.ServerMain
        -dig -digport 8000 -ontologies "."
   ```

2. **Launch DION2 server**: click on the file '$dion\_server.pl$' in the DION2 directory. If you encounter the global stack limit problem because of a big amount of test data, you should increase the size of the global stack. The windows users can edit the path setting of 'plwin.exe' in the file 'dionserver_bigGlobalStack.bat', then launch it.

The TELL and ASK request data can be copied into the TELL text area and the ASK text area respectively. After that, you can click on the buttons 'Tell' and 'Ask' to make the corresponding requests. The request data can also be posted from any application without using the DION2 testbed. That is useful if the test data exceeds the text area limit.

Chapter 4

Experiment

4.1 SEKT Ontology Learning Data

Among the most interesting new developments within the SEKT project one surely has to mention the efforts of the Text2Onto team in learning expressive ontologies. Often the potential of ontologies cannot be fully exploited, because information such as disjointness of concepts is not made explicit, and as a consequence, possible modeling errors remain undetected.

In Deliverable 3.3.3 [27] a number of learning features for disjointness axioms, and their use in a classifier, are described. The general idea is that there are different methods to estimate disjointness of two concepts, e.g. based on taxonomic overlap, semantic similarity, linguistic patterns etc., which are automatically combined by a machine learning classifier. Given a gold-standard, this classifier would basically be trained to predict whether an unseen pair of two concepts is disjoint or not.

Obviously, the problem of this approach is that a gold standard is needed, i.e. an agreed-upon collection of pairs of disjoint and non-disjoint concepts. In [27] the authors describe several experiments to construct such a gold-standard. Some of the most important findings from their perspective are that disjointness seems to be very difficult to understand for humans, and that it can be captured surprisingly well by automatic means. More surprising fact is that the agreement among the experts was not much higher than the agreement among the students.

Another finding of the experiments was that even disjointness axioms introduced by full agreement of all annotators can, if included in the ontology, lead to incoherence, i.e. logical contradictions. To investigate the cause of this contradiction we applied DION to debug the newly constructed ontologies.

4.1.1 Disjointness Statement in PROTON

The PROTON Ontology (PROTo ONtology) is an ontology which is developed in the scope of the SEKT Project[1]. For their experiments the authors of [27] constructed a representative data-set consisting of 2000 pairs of classes from PROTON, which they randomly assigned to 30 different people of two different levels expertise in ontology management. Each pair was tagged by 6 different people. Each of the annotators was given around 400 pairs along with natural language descriptions of the classes whenever those were available. Possible taggings for each pair were + (disjoint), (not disjoint) and ? (unknown). Based on the expertise of the annotators, and the required level of inter-annotator agreement several "gold standards" were developed. Deliverable 3.3.3 [27] discusses in detail the annotations of the different groups

4.1.2 Debugging Inconsistency in Extended PROTON

The first annotated data-set we focus on in this Deliverable is the one where a pair of concepts is only then considered to be disjoint when all 6 annotators agree, i.e. the data-set with disjointness statements of highest confidence. These disjointness statements are added to the PROTON ontology, an upper level ontology described in detail in SEKT Deliverable 1.8.1. The authors call the resulting ontology `proton_100_all.owl`, which stands for an extension of PROTON with disjointness statements where `all` annotators have to agree (100%).

Adding the disjointness statements results in a number of unsatisfiable concepts.

```
<unsatisfiableConcept ontology="proton_100_all.owl" number="3">
    <catom name="ns:Reservoir"/>
    <catom name="ns:Harbor"/>
    <catom name="ns:Canal"/>
</unsatisfiableConcept>
```

This means that there are three concepts, which are unsatisfiable: $Reservoir$, $Harbor$ and $Canal$. This is the standard output of any reasoner, i.e. Racer or KAON.

For these unsatisfiable concepts we calculate the minimal subsets of the ontology (MUPS) which retain the concept still unsatisfiable. Basically, this means that these sub-ontologies still contain a logical contradiction (which might point to an error). These MUPS are in the XML output of DION:

```
<MUPSs concept = "ns:Canal" number = "1" timecost = "0:3:11:252">
<mups>
  <disjoint>
```

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

```
    <catom name = "ns:Facility"/>
    <catom name="ns:WaterRegion"/>
  </disjoint>
  <impliesc>
    <catom name = "ns:Canal"/>
    <catom name ="ns:Channel"/>
  </impliesc>
  <impliesc>
    <catom name = "ns:Canal"/>
    <catom name = "ns:HydrographicStructure"/>
  </impliesc>
  <impliesc>
    <catom name = "ns:Channel"/>
    <catom name ="ns:WaterRegion"/>
  </impliesc>
  <impliesc>
    <catom name = "ns:HydrographicStructure"/>
    <catom name = "ns:Facility"/>
  </impliesc>
</mups>
</MUPSs>
```

and similarly for the other MUPS. From these MUPS we calculate the MIPS: remember, a MIPS is a minimal subset of the ontology which maintains incoherence, i.e. unsatisfiability of at least one unsatisfiable concept. These MIPS contain at least one ”error”, they are orthogonal to classical ”diagnoses”. They are not unique. In this example there are three different ones.

The three MIPS are

```
[[
axiom('ns:Facility_disjoint'),
axiom('ns:Reservoir_ns:HydrographicStructure_impliesc'),
axiom('ns:Reservoir_ns:Lake_impliesc'),
axiom('ns:HydrographicStructure_ns:Facility_impliesc'),
axiom('ns:Lake_ns:WaterRegion_impliesc')
],

[
axiom('ns:HydrographicStructure_ns:Facility_impliesc'),
axiom('ns:Facility_disjoint'),
axiom('ns:Harbor_ns:HydrographicStructure_impliesc'),
axiom('ns:Harbor_ns:WaterRegion_impliesc')
],

[
```

```
axiom('ns:Facility_disjoint'),
axiom('ns:Canal_ns:Channel_impliesc'),
axiom('ns:Canal_ns:HydrographicStructure_impliesc'),
axiom('ns:Channel_ns:WaterRegion_impliesc'),
axiom('ns:HydrographicStructure_ns:Facility_impliesc')
]].
```

Here each of the `axiom(...)` is an abbreviation for the following:

```
axiom('http://ns:Facility_disjoint') =
  <disjoint>
      <catom name="ns:Facility"/>
      <catom name="ns:WaterRegion"/>
  </disjoint>

axiom('ns:Reservoir_ns:HydrographicStructure_impliesc')=
  <impliesc>
      <catom name="ns:Reservoir"/>
      <catom name="ns:HydrographicStructure"/>
  </impliesc>

axiom('ns:Reservoir_ns:Lake_impliesc')=
 <impliesc>
      <catom name="ns:Reservoir"/>
      <catom name="ns:Lake"/>
  </impliesc>

etc.
```

So what do the MIPS tell us? Let's take again the first one:

```
[[
axiom('ns:Facility_disjoint'),
axiom('ns:Reservoir_ns:HydrographicStructure_impliesc'),
axiom('ns:Reservoir_ns:Lake_impliesc'),
axiom('ns:HydrographicStructure_ns:Facility_impliesc'),
axiom('ns:Lake_ns:WaterRegion_impliesc') ],
```

If we take a subset of the ontology (in a more DL like notation):

```
Facility ≠WaterRegion
Reservoir⊑HydrographicStructure
Reservoir⊑Lake⊑HydrographicStructure⊑Facility
Lake⊑WaterRegion
```

we have a minimal set of axioms making the ontology incoherent. Take one away, and the remaining ontology becomes coherent. This means that one of the axioms might be "incorrect". There are now 3 different of these MIPS.

But which one is the most likely candidate to be incorrect? The classical "diagnosis" way is to look at all the MIPS, and find the minimal hitting sets. But this is computationally impossible, so that we have a heuristic to find a small set of axioms that needs to be fixed to remove the logical contradiction. There are two different pinpoints:

```
[axiom('ns:Facility_disjoint')],
```

and

```
[axiom('ns:HydrographicStructure, ns:Facility_impliesc')]]
```

(these only have 1 element, but this is not necessarily so).

This means (e.g. for the first pinpoint): deleting the axioms in the pinpoint (here it is just one) makes your ontology logically correct, and there is almost always no smaller set of deletion axioms. So, in the spirit of Occam's razor we suggest that these axioms are the best axioms to be fixed for your ontology.

More concretely these are the axioms `disjoint(Facility,WaterRegion)` and `HydrographicStructure` $\sqsubseteq$ `Facility`.

According to this analysis, both axioms are erroneous with equal likelihood. So, for this particular set of unsatisfiable concepts (`reservoir, harbour, canal`) you now have two axioms, each of which could be erroneous, and fixing one of which will suffice to make your ontology consistent.

The ontology including PROTON and only disjointness axioms agreed by all annotators only contains 3 unsatisfiable concepts. The situation is much worse in the case when disjointness axioms are added to PROTON which were agreed upon by 3 ontology modeling experts (instead of 6 random annotators). In this case, an analysis using a debugging tool such as DION can even be more useful to determine the cause of a logical error.

On the other hand, since only one of these axioms was generated automatically, it might be reasonable to assume that this one is incorrect. Moreover, confidence or relevance information acquired during the ontology learning process might help to select the axiom, which is most likely to be incorrect[2].

4.1.3 Summary

Learning expressive statements about ontologies, and extending existing ontologies with expressive statements (such as disjointness statements) is a difficult issue, which can eas-

[2] http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id=1007

ily lead to inconsistencies, even if the utmost care is taken to ensure the highest quality of new axioms.

This makes the existence of automatic debugging support existential whenever the expressiveness of an ontology is increased. This holds both for the case of carefully selected extensions (such as shown in the above example), and even more so when the extensions are learned automatically. Combining ways of automatically learning expressive ontology axioms with debugging is one of foremost challenges for the proposed methods of this deliverable.

4.2 SEKT Legal Ontologies

The aim of SEKT is to develop and exploit semantically-based knowledge technologies in order to support document management, content management, and knowledge management in knowledge intensive workplaces. Specifically, SEKT aims at designing appropriate utilities to users in three main areas: digital libraries, the engineering industry, and the legal domain, providing them with quick access to the right pieces of information at the right time. SEKT legal otologies are developed for the SEKT legal case study. More details regarding the legal case study can be found in SEKT deliverable D10.2.1[4].

In the legal case study, the accomplished tasks so far provide both the quantitative and qualitative data necessary to assess both the context of users, (newly recruited Spanish judges), and their specific needs with regard to the technology under development. In particular, these data give an insight on institutional, organizational, and individual constraints that could either facilitate or block the introduction of SEKT technologies in judicial units. From this set of data, the Ontology of Professional Judicial Knowledge is built.

The data (nearly 800 competency questions) gathered from judicial interviews by the legal case study, allows the construction of an ontology based on professional judicial knowledge. Knowledge derived from the daily practice at courts. The Ontology of Professional Judicial Knowledge (OPJK) developed by the legal case study team has been learnt from scratch out of nearly 400 competency questions and has, currently, nearly 104 classes and 567 instances. The following top domain concepts have been identified: $Acto\_Procesal$, $Organo\_Judicial$, $Calificacion\_Jurdica$, $Documento\_Procesal$, $Fase\_Procesal$, $Jurisdiccion$, $Proceso\_Judicial$, $Profesion\_Jurdica$, $Rol\_Procesal$, $Rol\_Familiar$ and $Sancion$.

The OPJK ontology constitutes the core of the legal case study within SEKT, as it links natural language questions (formulated by newly recruited Spanish judges) with a repository of frequently asked questions (FAQs) with their corresponding answers provided by more experienced judges. To this aim, OPJK contains definitions for the most relevant judicial terms in the professional knowledge of law professionals with respect to this set of FAQs. As this is highly specialized knowledge producing a new version of the

OPJK is a difficult task. In practice, a team of legal experts meets in regular intervals to decide on the relevant changes to the OPJK where the decision is taken w.r.t. the questions in the FAQs.

The current version of the OPJK ontolgoy is specified in RDF/RDFS data format. It does not cause the inconsistency problem because of its limited language expression. To enhance the semantics of the SEKT legal ontologies, we convert the OPJK data from RDF/RDFS into DIG first, then add disjointness statements for sibling concepts into OPJK. This procedure is called semantic enhancement in [23]. We obtain 26 disjointness statements by an automatic computation of sibling relations.

Here are some disjointness statements of OPJK in DIG format:

```
<disjoint>
  <catom name="Situacion"/>
  <catom name="Evento"/>
  <catom name="Acto"/>
</disjoint>

<disjoint>
  <catom name="Fase_de_Juicio_oral"/>
  <catom name="Fase_de_ejecucion_penal"/>
  <catom name="Fase_de_instruccion"/>
</disjoint>

<disjoint>
  <catom name="Fase_de_alegaciones"/>
  <catom name="Fase_de_prueba"/>
  <catom name="Fase_de_Conclusiones"/>
</disjoint>

<disjoint>
  <catom name="Conyuge"/>
  <catom name="Progenitor"/>
</disjoint>

<disjoint>
  <catom name="Estado_Temporal"/>
  <catom name="Estado_Proceso"/>
  <catom name="Estado_Mental"/>
  <catom name="Estado_Fisico"/>
</disjoint>

<disjoint>
  <catom name="Fase_Procesal_Penal"/>
  <catom name="Fase_Procesal_Civil"/>
</disjoint>

<disjoint>
  <catom name="Administracion_Publica"/>
  <catom name="Empresa"/>
```

```
</disjoint>

<disjoint>
  <catom name="Mayoraa_de_Edad"/>
  <catom name="Minoraa_de_Edad"/>
</disjoint>
```

Adding those disjoint axioms on OPJK, we obtain an inconsistent OPJK ontology in which there exist thres unsatisfiable concepts: $Profesion\_Juradica\_Liberal$, $Funcionario\_Judicial$, and $Profesion\_Juradica$. We use this inconsistent OPJK ontology as a test data on DION2.

DION2 finds the following MUPS for those unsatisfiable concepts:

```
<MUPSs concept="Funcionario_Judicial" number="1" timecost="0:0:0:460">
  <mups>
<disjoint>
        <catom name="Rol_Familiar"/>
        <catom name="Rol_Negocio_Juradico"/>
        <catom name="Rol_Procesal"/>
        <catom name="Rol_Profesional"/>
      </disjoint>
<impliesc>
        <catom name="Funcionario_Judicial"/>
        <catom name="Profesion_Juradica"/>
      </impliesc>
<impliesc>
        <catom name="Profesion_Juradica"/>
        <catom name="Rol_Procesal"/>
      </impliesc>
<impliesc>
        <catom name="Profesion_Juradica"/>
        <catom name="Rol_Profesional"/>
      </impliesc>
   </mups>
</MUPSs>


<MUPSs concept="Profesion_Juradica_Liberal" number="1" timecost="0:0:0:451">
  <mups>
     <disjoint>
       <catom name="Rol_Familiar"/>
       <catom name="Rol_Negocio_Juradico"/>
       <catom name="Rol_Procesal"/>
       <catom name="Rol_Profesional"/>
    </disjoint>
    <impliesc>
      <catom name="Profesion_Juradica_Liberal"/>
      <catom name="Profesion_Juradica"/>
    </impliesc>
    <impliesc>
```

```
      <catom name="Profesion_Juradica"/>
      <catom name="Rol_Procesal"/>
    </impliesc>
    <impliesc>
      <catom name="Profesion_Juradica"/>
      <catom name="Rol_Profesional"/>
    </impliesc>
  </mups>
</MUPSs>

<MUPSs concept="Profesion_Juradica" number="1" timecost="0:0:0:269">
  <mups>
    <disjoint>
      <catom name="Rol_Familiar"/>
      <catom name="Rol_Negocio_Juradico"/>
      <catom name="Rol_Procesal"/>
      <catom name="Rol_Profesional"/>
    </disjoint>
  <impliesc>
    <catom name="Profesion_Juradica"/>
    <catom name="Rol_Procesal"/>
  </impliesc>
  <impliesc>
     <catom name="Profesion_Juradica"/>
     <catom name="Rol_Profesional"/>
   </impliesc>
  </mups>
</MUPSs>
```

We can see that the MUPS for the concept $Profesion\_Juradica$ is a subset of the MUPS of two other unsatisfiable concepts. Thus we can consider only the MUPS for the unsatisfiable concept $Profesion\_Juradica$. We use $disjoint(Rol\_Familiar)$, $impliesc_1$ and $impliesc_2$ to denote those three axioms in the MUPS respectively. The MIPS of the OPJK ontology is the set $\{disjoint(Rol\_Familiar), impliesc_1, impliesc_2\}$. Since there exists only one MIPS for the OPJK ontology, naturally the pinpoint for the OPJK debugging consists of only those three axioms. We observe that removing the axiom $disjoint(Rol\_Familiar)$, i.e., the disjoint axiom on the concepts $Rol\_Familiar$ and $Rol\_Procesal$ and $Rol\_Profesional$ can restore the consistency of the OPJK ontology sufficiently.

4.3 Preprocessing of Inconsistent Ontologies

For an inconsistent ontology, DION2 can find pinpoints show which axioms should be removed from the inconsistent ontology to restore the consistency. Sometimes it is more useful to remove parts of axioms instead of whole axioms. For example, for a subsumption axiom $C \sqsubseteq D_1 \wedge D_2$ which appears in a pinpoint, we would not remove the whole

axiom $C \sqsubseteq D_1 \sqcap D_2$ from an inconsistent ontology, but just remove part of axiom, say $C \sqsubseteq D_1$, from the ontology. The removal can sufficiently disqualify the whole axiom sufficiently. This approach allows us to achieve more fine-grained debugging. In order to achieve it, DION2 will do preprocessing on an inconsistent ontology when it is loaded into the system.

Under this preprocessing procedure, an axiom is converted into several sub-axioms, which are semantically equal to the original axiom. The following are the conversion rules which are used in DION2. An axiom which has the form of the left hand part of a conversion rule is replaced with a set of axioms by the right hand part of the conversion rule. Namely, when an axiom is loaded, it is checked whether or not it can be converted by one of the following rules. If yes, then the axiom is replaced with a set of new axioms. Those newly created axioms are checked further to see whether or not they can be applied by one of the conversion rules further until there are no new axioms can be created by the conversion.

- **Conjunction Decompostion**
  $C \sqsubseteq (D_1 \sqcap D_2 \ldots D_n) \Rightarrow \{C \sqsubseteq D_1, C \sqsubseteq D_2, \ldots, C \sqsubseteq D_n\}$.
- **Equivalence Decomposition**
  $C = D \Rightarrow \{C \sqsubseteq D, D \sqsubseteq C\}$.
- **Existence Conversion**
  $C \sqsubseteq \exists R.(E_1 \sqcap E_2 \ldots \sqcap E_n) \Rightarrow \{C \sqsubseteq \exists R.E, E = (E_1 \sqcap E_2 \ldots \sqcap E_n)\}$
  where E is a new named concept.
- **Universal Conversion**
  $C \sqsubseteq \forall R.(E_1 \sqcap E_2 \ldots \sqcap E_n) \Rightarrow \{C \sqsubseteq \forall R.E, E = (E_1 \sqcap E_2 \ldots \sqcap E_n)\}$
  where E is a new named concept.

For an axiom, only one conversion rule can be used each time, because the left hand side of conversion rules are different each other. It is also easy to see that the conversion algorithm can terminate.

Taking our working example which is discussed in Chapter 2, DION2 finds the following two pinpoints without the preprocessing:

$$\{A_4 \sqsubseteq \forall s.B \sqcap C, A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3\}$$
$$\{A_4 \sqsubseteq \forall s.B \sqcap C, A_2 \sqsubseteq A \sqcap A_4\}$$

With the preprocessing, DION2 finds the three more fine-grained pinpoints:

$$\{A_4 \sqsubseteq \forall s.B, A_1 \sqsubseteq A_2\}$$
$$\{A_4 \sqsubseteq \forall s.B, A_1 \sqsubseteq \neg A\}$$
$$\{A_4 \sqsubseteq \forall s.B, A_2 \sqsubseteq A\}$$

Chapter 5

Discussion and Conclusions

5.1 Related Work

In this section, we briefly discuss the most important related work in the literature. In our earlier conference publications [22] we were the first to propose the framework for debugging and diagnosing of terminologies that is defined in Chapter 2. There we also coined the term *pinpointing* as a means of reducing a logically incorrect terminology to a smaller one, from which a modeling error could be more easily detected by a human expert. In [24] we grounded notions of MIPS and MUPS in the well-established theory of model-based diagnosis [20].

The area of debugging inconsistent ontologies has received much attention since the publication of [22]. In this section we do not aim to give a general literature survey, but discuss some of the most influential pieces of work, in particular the work by the MindSwap group, and the work based on belief revision.

5.1.1 Debugging in the DL community

The MindSwap group at the University of Maryland has done significant work in this area, culminating in the recent thesis of Kalyanpur [14]. The work investigates two different approaches, one based on modifying the internals of a DL reasoner (the so-called "glass box" approach), and one based on using an unmodified external reasoner (the "black box" approach).

The glass box approach is closely related to our work of the top-down approach, and (just as our work) is based on the techniques in [1]. The work deals with OWL-Lite, except for maximum cardinality roles, and is efficient since it avoids having to do full tableau saturation (details are in [16]). The work in [1] is particularly noteworthy: Although that paper is about a different topic (computing extensions for a certain class of default Description Logics), it turns out that one of the algorithms is very similar to the

one described in section 2. The main difference to Baader et. al's work is that they consider ABoxes instead of TBoxes, and the purpose of the algorithm (computing default extensions vs. computing diagnoses).

The black box approach (i.e.. detecting inconsistencies by calling an unmodified external DL reasoner) is based on Reiter's Hitting Set algorithm (similar to our work in [24]), and also closely related to a proposal of Friedrich et al. [7] who (as we do in section 2.2.1) bring the general diagnostic theories from [20] to bear for diagnosing ontologies. An interesting difference with our work is that Friedrich et. al use generic diagnoses software. As we do in our bottom-up method, they use a DL reasoner as oracle.

Kalyanpur also proposes a method for "axiom pinpointing"[1], which rewrites axioms into smaller ones, and then debugs the resulting ontology after rewriting, with the effect that a more precise diagnosis is obtained. Early results have been reported in [15]

A second pinpointing technique called "error pinpointing" by Kalyanpur is similar to what we call pinpointing here. Interestingly, Kalyanpur has performed user studies which reveal that a combination of axiom pinpointing (i.e.. breaking large axioms up into smaller ones) and error pinpointing (i.e. finding the errors which lie at the root of a cascading chain of errors) together seems to be the cognitively most efficient support for users.

Finally, a significant extension to our work in [22] was published in [18], where the authors extend our saturation based tableau calculus with blocking conditions, so that general TBoxes can be handled.

5.1.2 Belief revision

Much of the work in the belief revision community over the past twenty years has focused on dealing with inconsistency, and significant advances have been made [10]. Nevertheless, there are significant differences which cause this work to be not directly applicable to ontology revision. First of all, most of the work on belief revision is phrased in terms of a so called "belief base", a deductively closed set of formulae. Much of the interest in dealing with inconsistent ontologies is to deal with sets of axioms that are not deductively closed, and on which deduction has to be performed in order to find out inconsistencies and their causes. Furthermore, theories of belief revision typically assume a preference ordering among all models of a belief base, representing an order of implausibility among all situations. Such an approach is also taken in [17] which imposes a stratification on the knowledge base, and employing this stratification to select a suitable repair.

There has been work on belief revision that does not rely on deductively closed belief bases, and hence is more relevant to our work. One such example is [9], from whom we have taken the notion of a syntactic relevance function. (Such a syntactic relevance func-

[1] A different use of the word pinpointing from our use in section 2.2.2, and even from the equivalent term "axiom pinpointing" in [22].

tion only makes sense when abandoning the notion of deductively-closed belief bases, since an immediate consequence of working with deductively closed belief bases is that equivalent formulae should be treated equally, a.k.a. the principle of irrelevance of syntax). Our work in section 3 can be seen as a specialization to ontologies of the general framework presented in [9].

5.2 Conclusion

We present a formal characterization for debugging and diagnosis (briefly repeated from earlier publications); we define algorithms for all tasks described in our debugging framework; we study the effectiveness of our proposal in a realistic setting on life-size terminologies; we perform a set of controlled experiments to analyst the computational properties of the debugging problem and our different algorithms for solving it in [25].

We have implemented DION2, a new prototype of DION as its extension, based on the bottom-up approach of inconsistent ontology debugging. The new functionalities of DION2 include multiple platform support, integration with KAON2, fine-grained debugging by preprocessing. In this document, we also report two experiments of DION2 with SEKT case study data: one from SEKT ontology learning data and one from SEKT legal ontologies. Those preliminary tests show that how DION2 can be used for inconsistent ontology diagnosis and repair in realistic application scenarios.

Bibliography

[1] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *J. Automated Reasoning*, 14:149–180, 1995.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Sean Bechhofer, Ralf Möller, and Peter Crowther. The dig description logic interface. In *International Workshop on Description Logics (DL2003)*. Rome, September 2003.

[4] P. Casanovas, M. Poblet, N. Casellas, J-J. Vallbé, F. Ramos, V.R. Benjamins, M. Blázquez, J. Contreras, and J. Gorronogoitia. Legal scenario. case study intelligent integrated decision support for legal professionals. Project Report Report D10.2.1, SEKT, 2005.

[5] Luca Console and Oskar Dressler. Model-based diagnosis in the real world: Lessons learned and challenges remaining. In *Proceedings of the sixteenth International Joint Conference on Artificial Intelligence, IJCAI'99*, pages 1393–1400. 1999.

[6] Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 32–43. ACM, 2003.

[7] Gerhard Friedrich and Kostyantyn M. Shchekotykhin. A general diagnosis method for ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2005.

[8] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiters theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[9] S. O. Hansson and R. Wassermann. Local change. *Studia Logica*, 70(1):49–76, 2002.

[10] S.O. Hansson. *A Textbook of Belief Dynamics*. Kluwer Academic Publisher, Dordrecht, 1999.

[11] Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'05*, 2005.

[12] Zhisheng Huang and Frank van Harmelen. Reasoning with inconsistent ontologies: Evaluation. Deliverable D3.4.2, SEKT, 2005.

[13] Zhisheng Huang and Cees Visser. Extended dig description logic interface support for prolog. Deliverable D3.4.1.2, SEKT, 2004.

[14] A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, Univ. of Maryland, June 2006.

[15] A. Kalyanpur, B. Parsia, B. Cuenca-Grau, and E. Sirin. Beyond axioms: Fine-grained justifications for arbitrary entailments in owl-dl. In B. Parsia, U. Sattler, and D. Toman, editors, *DL*, 2006.

[16] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable concepts in owl ontologies. *Journal of Web Semantics*, 3(4), 2005.

[17] Th. Meyer, K. Lee, and R. Booth. Knowledge integration for description logics. In *AAAI*, pages 645–650, 2006.

[18] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic alc. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI'06*, 2006.

[19] B. Nebel. Terminological reasoning is inherently intractable. *AI*, 43:235–249, 1990.

[20] R. Reiter. A theory of diagnosis from first principles. *Artif. Intelligence*, 32(1):57–95, 1987.

[21] S. Schlobach. Diagnosing terminologies. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 670–675, 2005.

[22] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.

[23] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *ESWC*, pages 226–240, 2005.

[24] Stefan Schlobach. Diagnosing terminologies. In *Proceedings of AAAI'2005*, pages 670–675, 2005.

[25] Stefan Schlobach, Ronald Cornet, and Zhisheng Huang. Inconsistent ontology diagnosis. Project Report D3.6.2, SEKT, 2006.

[26] Stefan Schlobach and Zhisheng Huang. Inconsistent ontology diagnosis: Framework and prototype. Project Report D3.6.1, SEKT, 2005.

[27] Johanna Völker, Denny Vrandecic, and York Sure. Data-driven change discovery. Deliverable D3.3.3, SEKT, 2007.