Facultad de Informática
Universidad Politécnica de Madrid

European Master In Computational Logic

Master Thesis

An AntiPattern-Based OWL Ontology Debugging Tool

by

Mohamad Fauzan Tahwil

Supervisor: Prof. Oscar Corcho García
Co-Supervisor: Dr. Catherine Roussey

Madrid, June 2010

As Sunnah : Utlubul ilma minal mahdi ilal lahdi - Seek knowledge from the cradle to the grave

Contents

Abstract

Debugging an OWL ontology manually is difficult and a time-consuming task, even for ontology engineering and knowledge representation experts. Some debugging tools are very dependent on to the reasoners that may take several hours to detect inconsistencies. Current debugging tools do not provide enough information. None of these tools can detect modeling errors and suggest a better form of expression. By continuing previous research on antipatterns, we want to make an effort in this thesis to build Apero, a Protégé plug-in that offers an ontology debugging tool based on the use of antipatterns. Apero is able to detect inconsistencies without the use of a reasoner, shows possible modeling errors and suggests an alternative form of expression but more accurate for encoding the same knowledge.

Acknowledgements

The first thanks goes to my supervisor, Prof. Oscar Corcho García, for all his support and guidance. He was always ready (even during holiday season, week ends, and nights) to help me with all my difficulties. I really appreciate his support in this thesis as well as in practical work.

I would like to express my gratitudes to all my professors in UPM and TUD. Special thanks go to Prof. Steffen Hölldobler for giving me opportunity in this master program, Prof. Francisco Bueno to support my education in UPM.

I would like to express my thanks to all my friends in UPM and TUD. To Catherine for always providing me whatever sources I need in order to complete this work and giving me feedback. To Evgeny, Seif and Milka for sharing fun time with me in Dresden. Thank you to my flat mates (Luis, Ruben and Sinan) for being my family here. Also Freddy, Irene and Anton have been keeping me to speak Indonesian Language and supporting my thesis.

To my friend and lecturer, Prof. Lim Yohanes Stefanus from University of Indonesia. Thanks for supporting me in the very first place so that I am able to get scholarship in this master program.

To the people I miss during my master study, my father, mother and sisters, for giving me big support to my success.

To my beloved wife who always accompanies me from loneliness although she is be in different place. Her kindness, support and patience make me see the future more beautiful.

This thesis is supported by Erasmus Mundus scholarship financial grant.

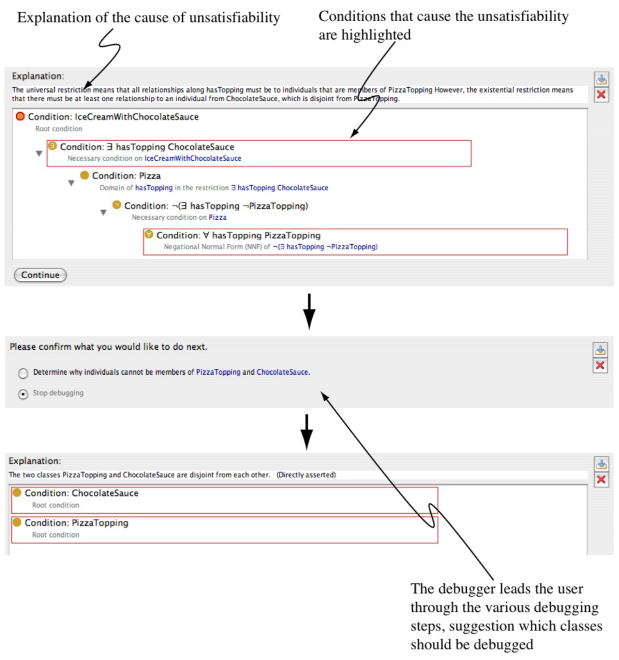
Chapter 1

Introduction

In recent years there has been a considerable amount of interest in the area of debugging and repair of OWL ontologies. The process of debugging ontology is as important as debugging program code. They are vital to get rid of faults. In ontology debugging, the faults are marked as undesirable entailments. In particular, the entailment that a concept (class) is unsatisfiable is almost always undesired. Nevertheless, undesirable entailments are not limited to unsatisfiable classes. Other undesirable entailments may be caused by unintended and conflicting to the modeler's understanding of the domain, for instance certain subclass relation between classes. Ontology debugging is the process of finding the causes of an undesirable entailment, understanding these causes, and modifying the ontology so that there is no longer the undesirable entailment [8].

It is a tremendously hard task discovering the cause of errors or faults. DL (Description logic) reasoners can only supply lists of unsatisfiable classes when checking satisfiability (consistency). They present no extra explanation about why a class is unsatisfiable. An unsatisfiable class could be a derived or root unsatisfiable class. A derived unsatisfiable class is a class whose satisfiability depends on another class, otherwise it is classified to a root unsatisfiable class. We have to examine and fix root unsatisfiable classes first. For instance, A is unsatisfiable class, $B \sqsubseteq A$ and $C \sqsubseteq \exists R.A$ imply B and C as derived unsatisfiable classes.

Users manually do the process of ontology debugging to find the reason of the unsatisfiability of the class. However, users do not always have enough experience with DL. Thus, it will be a hard task for them without adequate supporting tools. It will be more complicated when the ontology is large and complex. Even experienced ontology engineers will have difficulty to find the causal error. The ability to provide this causal error can be a measurement of powerfulness of DL reasoners [23].


Explanation of the cause of unsatisfiability
Conditions that cause the unsatisfiability are highlighted
The debugger leads the user through the various debugging steps, suggestion which classes should be debugged
Continue

Explanation:
The universal restriction means that all relationships along hasTopping must be to individuals that are members of PizzaTopping However, the existential restriction means that there must be at least one relationship to an individual from ChocolateSauce, which is disjoint from PizzaTopping.

Condition: IceCreamWithChocolateSauce
Root condition
Condition: ∃ hasTopping ChocolateSauce
Necessary condition on IceCreamWithChocolateSauce
Condition: Pizza
Domain of hasTopping in the restriction ∃ hasTopping ChocolateSauce
Condition: ¬(∃ hasTopping ¬PizzaTopping)
Necessary condition on Pizza
Condition: ∀ hasTopping PizzaTopping
Negational Normal Form (NNF) of ¬(∃ hasTopping ¬PizzaTopping)

Please confirm what you would like to do next.
○ Determine why individuals cannot be members of PizzaTopping and ChocolateSauce.
[✓ Stop debugging]

Explanation:
The two classes PizzaTopping and ChocolateSauce are disjoint from each other. (Directly asserted)
Condition: ChocolateSauce
Root condition
Condition: PizzaTopping
Root condition

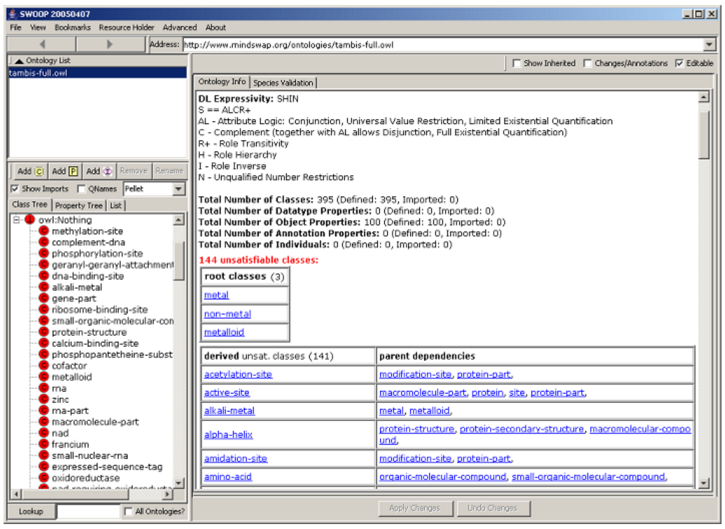
Figure 1.1: Screenshot of Protégé OWL Debugger

When we are focusing on DL formalization and OWL implementation, there are several options of ontology debugging tool, that have proven their effectiveness in different domains, as follows:

- Protégé OWL Debugger
  Protégé OWL Debugger [9] is a tool to help people debug OWL-DL ontologies. As its name suggests, it is incorporated in Protégé as a plug-in. It provides conditions that cause a class be unsatisfiable. It also guides the user through the various debugging steps. Unfortunately, it requires to start the OWL reasoner (e.g RACER or FaCT++) and classify the ontology first. This process will become a time-consuming task if we face a very large and complex ontology. The Screenshot of this tool is shown with the Figure 1.1. *

- SWOOP
  SWOOP [15] is a standalone tool to create, edit, and debug OWL ontologies. Swoop uses a DL reasoner (e.g. Pellet) to determine which named concepts in the ontology are unsatisfiable. Thus, it has the same problem than the

*The source is available at *http://www.co-ode.org/downloads/owldebugger/*



DL Expressivity: SHIN
S == ALCR+
AL - Attribute Logic: Conjunction, Universal Value Restriction, Limited Existential Quantification
C - Complement (together with AL allows Disjunction, Full Existential Quantification)
R+ - Role Transitivity
H - Role Hierarchy
I - Role Inverse
N - Unqualified Number Restrictions

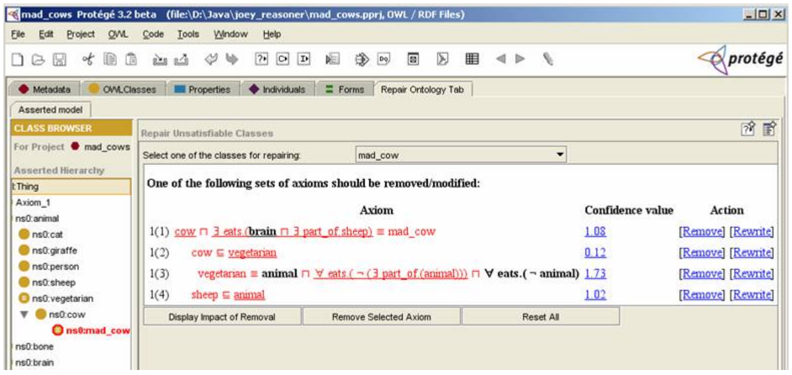
Total Number of Classes: 395 (Defined: 395, Imported: 0)
Total Number of Datatype Properties: 0 (Defined: 0, Imported: 0)
Total Number of Object Properties: 100 (Defined: 100, Imported: 0)
Total Number of Annotation Properties: 0 (Defined: 0, Imported: 0)
Total Number of Individuals: 0 (Defined: 0, Imported: 0)

144 unsatisfiable classes:

root classes (3)
metal
non-metal
metalloid

derived unsat. classes (141)	**parent dependencies**
acetylation-site	modification-site, protein-part,
active-site	macromolecule-part, protein, site, protein-part,
alkali-metal	metal, metalloid,
alpha-helix	protein-structure, protein-secondary-structure, macromolecular-compound,
amidation-site	modification-site, protein-part,
amino-acid	organic-molecular-compound, small-organic-molecular-compound,

Figure 1.2: Screenshot of SWOOP



Select one of the classes for repairing: mad_cow

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	$\text{cow} \sqcap \exists\,\text{eats}.(\textbf{brain} \sqcap \exists\,\text{part\_of}.\text{sheep}) \equiv \text{mad\_cow}$	1.08	[Remove] [Rewrite]
1(2)	$\text{cow} \sqsubseteq \text{vegetarian}$	0.12	[Remove] [Rewrite]
1(3)	$\text{vegetarian} \equiv \textbf{animal} \sqcap \forall\,\text{eats}.(\neg(\exists\,\text{part\_of}.(\text{animal}))) \sqcap \forall\,\textbf{eats}.(\neg\,\textbf{animal})$	1.73	[Remove] [Rewrite]
1(4)	$\text{sheep} \sqsubseteq \text{animal}$	1.02	[Remove] [Rewrite]

Figure 1.3: Screenshot of RepairTab

Protégé OWL Debugger. The Screenshot of this tool is shown with the Figure 1.2. [†]

- RepairTab
  RepairTab [16] is a Protégé plugin that suggests alternatives to resolve the identified inconsistencies. It shows those entailments that would be lost if the recommended solution was applied. However, the current solution is restricted to remove part from the existing axioms or to replace a class by one of its super classes. The Figure 1.3 is a screenshot of RepairTab with Mad Cow ontology as example.

Reviewing the previous research on antipattern [4], we concentrate our study

[†]The source is available at *http://www.mindswap.org/2005/debugging/*

on real ontologies that have been developed by domain experts [‡], who are not necessarily too familiar with DL, and hence can misuse DL constructors and misunderstand the semantics of some OWL expressions which may bring us to an unwanted unsatisfiability classes. For instance, HydrOntology is a medium-sized OWL ontology (165 classes) developed by a domain expert in the area of Hydrology [3]. The first version of this ontology has a total of 114 unsatisfiable classes.

To find the reasons for their unsatisfiability, domain experts find it difficult to understand the information provided by the debugging systems used ([9], [15]) on root unsatisfiability class. In addition, sometimes during the debugging process, the generation of justification for unsatisfiability took several hours. This made these tools hard to use. It confirms the result described in [20]. As a result, we found out that in several occasions the domain experts were just changing axioms from the original ontology in a somehow random way. They even change the intended meaning of the definitions instead of correcting errors in their formalization.

By using this ontology and several other real ontologies, in this thesis we identify common unsatisfiability-leading patterns used by the domain experts when implementing OWL ontologies. There will be also a solution that can be used by domain experts to debug their ontology. At the end, we provide some hints on how to organize the iterative ontology debugging process using a combination of debugging tools and patterns.

This thesis is organized as follows:

1. Chapter 2 presents state of the art and background theory of related technologies we use in our work.
2. Chapter 3 presents the objectivity of our work, including the assumption and limitations.
3. Chapter 4 presents OPPL-based antipattern.
4. Chapter 5 presents Lint-based antipattern.
5. Chapter 6 presents Apero Plugin, including its analysis, design and implementation process.
6. Chapter 7 presents the evaluation results of Apero plugin.
7. Chapter 8 presents the conclusions on our work and some future works.

[‡]Domain expert is a person with special knowledge or skills in a particular area

Chapter 2

State of the Art

This chapter discusses the latest technology that we use or that we refer to in this thesis.

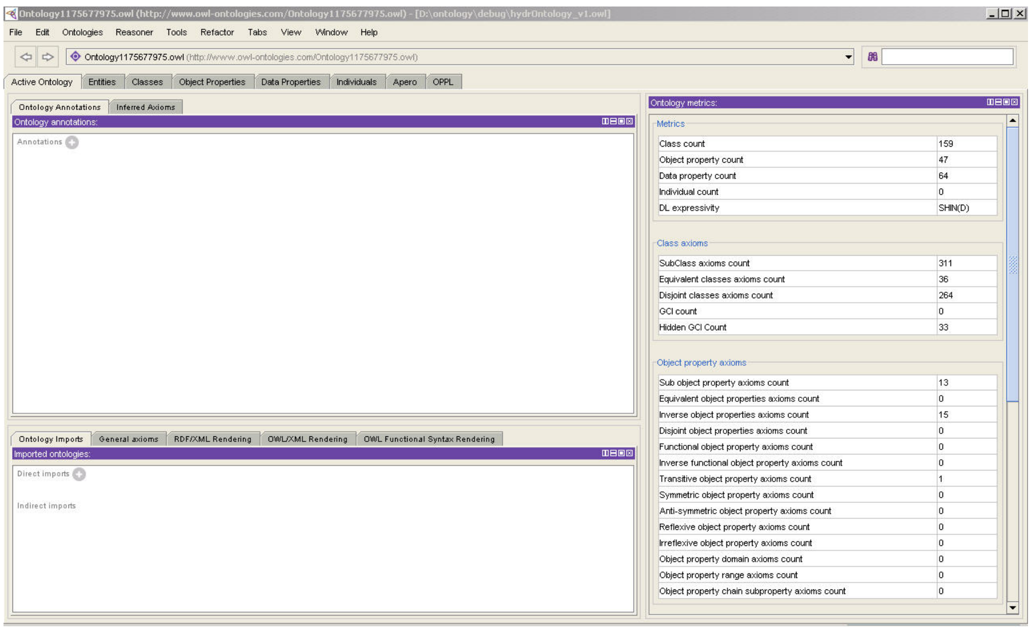
2.1 OWL and Protégé

OWL is the most recent development in standard ontology languages, endorsed by the World Wide Web Consortium (W3C) to promote the Semantic Web vision. ”An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms” [13].

OWL has three sub languages as follows [13]:

1. OWL Lite.
   OWL Lite has a classification hierarchy and simple constraint features. For instance, cardinality constraints only allow cardinality values of 0 or 1.

2. OWL DL.
   OWL DL has maximum expressiveness without losing computational completeness and decidability of reasoning systems. Completeness means that all entailments are guaranteed to be computed, whereas decidability means that all computations will finish in finite time. OWL DL is so named due to its correspondence with description logics, a field of research that has studied a particular decidable fragment of first order logic.


Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl) - [D:\ontology\debug\hydrOntology_v1.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals Apero OPPL
Ontology Annotations Inferred Axioms
Ontology annotations:
Annotations
Ontology Imports General axioms RDF/XML Rendering OWL/XML Rendering OWL Functional Syntax Rendering
Imported ontologies:
Direct imports
Indirect imports
Ontology metrics:

Metrics

Class count	159
Object property count	47
Data property count	64
Individual count	0
DL expressivity	SHIN(D)

Class axioms

SubClass axioms count	311
Equivalent classes axioms count	36
Disjoint classes axioms count	264
GCI count	0
Hidden GCI Count	33

Object property axioms

Sub object property axioms count	13
Equivalent object properties axioms count	0
Inverse object properties axioms count	15
Disjoint object properties axioms count	0
Functional object property axioms count	0
Inverse functional object property axioms count	0
Transitive object property axioms count	1
Symmetric object property axioms count	0
Anti-symmetric object property axioms count	0
Reflexive object property axioms count	0
Irreflexive object property axioms count	0
Object property domain axioms count	0
Object property range axioms count	0
Object property chain subproperty axioms count	0

Figure 2.1: Screenshot of Protégé

3. OWL Full.
   OWL Full has maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For instance, a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

Protégé is a free and open-source software that provides a user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats [11]. The GUI (Graphical User Interface) of Protégé is shown on the Figure 2.1.

There are several features that distinguish Protégé from other ontology editing tools. Protégé has all of the following features that cannot be found on other tools:

- Intuitive and user-friendly GUI.
- Extensible plug-in architecture. It is easy to extend Protégé with plug-ins designed for our domain and task.

Protégé supports OWL. It also has an extension (Protégé OWL editor) that enables us to load and save ontologies and execute description logic classifiers. Protégé can also be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

Axiom is a definition that associates class and property identifiers with specifications of their characteristics, and to give other logical information about classes and properties [13]. Example of an axiom is "AnimalLover *equivalentTo* Person *and* hasPet min 3"

Syntax of OWL in Protégé uses Manchester OWL syntax. We will find the use of OWL syntax itself in the owl file. Because of compactness, we will use DL symbol as a tool to represent an OWL expression. The table 2.1 shows the most common constructor in OWL, DL and Manchester OWL syntax *. Protégé provides OWL API that enables developer to access all axioms and manipulate them.

Table 2.1: Mapping of Syntax

Constructor	OWL	DL Symbol	Manchester OWL Syntax Keyword	Example
Negation	*complementOf*	$\neg C$	*not*	*not* Child
Intersection	*intersectionOf*	$C_1 \sqcap ... \sqcap C_n$	*and*	Doctor *and* Female
Union	*unionOf*	$C_1 \sqcup ... \sqcup C_n$	*or*	Male *and* Female
Universal Restriction	*allValuesFrom*	$\forall R.C$	*only*	hasSibling *only* woman
Existential Restriction	*someValuesFrom*	$\exists R.C$	*some*	hasChild *some* man
Maximum Cardinality	*maxCardinality*	$\leq nR.C$	*max*	hasChild *max* 3
Exact Cardinality	*exactCardinality*	$= nR.C$	*exactly*	hasChild *exatcly* 3
Minimum Cardinality	*minCardinality*	$\geq nR.C$	*min*	hasChild *min* 3
Spesific Value	*hasValue*	$\exists$ R.x	*value*	hasPlayer *value* zidane
Subclass	*subClassOf*	$C_1 \sqsubseteq$	*subClassOf*	Male *subClassOf* Human
Equivalence	*equivalenClass*	$C_1 \equiv C_2$	*equivalentTo*	AnimalLover *equivalentTo* Person *and* hasPet min 3
Disjointness	*disjointWith*	$C_1 \sqcap C_2 \sqsubseteq \bot$	*disjointWith*	Dog *disjointWith* Cat

Reasoner is a piece of software that is able to detect inconsistencies, find subclasses and detect trivially satisfiable class. Depending on the expressiveness of OWL (OWL Lite or DL), a reasoner is either OWL Lite reasoner or DL-reasoner. DL-reasoner also covers the Lite one. We exclude reasoner for OWL Full in this thesis because of decidability reasons.

Protégé provides facility to use reasoner. Reasoner can be access from menu [Reasoner >Classify] if a reasoner has been selected or [Reasoner >*Reasoner-Name*] (see the Figure 2.2). After reasoning process is complete, Protégé will mark every unsatisfiable class with red color.

2.2 OPPL

OPPL (Ontology Pre-Processor Language) is an abstract formalism that allows manipulating OWL ontologies [14]. The version of the Ontology Pre-Processor Language (OPPL) described here is the successor of the initial effort presented in [7]. OPPL provides a plug-in for Protégé and API (Application Programming

*Full list available in reference specs and in the Quick Reference Guide. (see *http://www.w3.org/2007/OWL/refcard)*

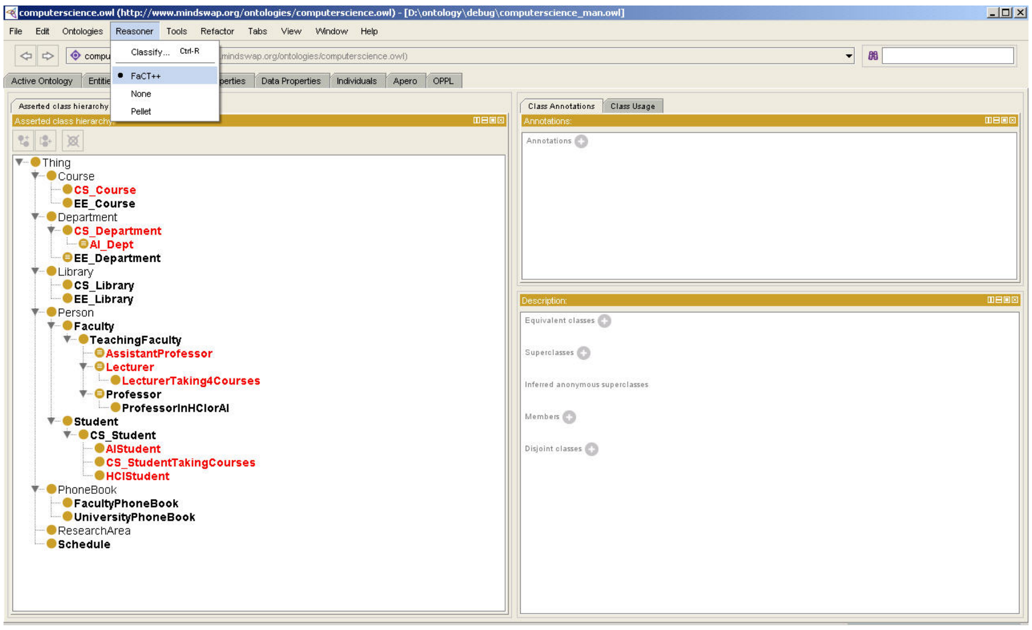


Figure 2.2: Reasoner in Protégé

Interface). As a plug-in, users can build and run OPPL statements. Meanwhile, we can use the API of OPPL to run OPPL statements from our own application.

A generic OPPL statement will look as follows:

```
SELECT Axiom,....,Axiom
BEGIN
ADD | REMOVE Axiom
...
ADD | REMOVE Axiom
END
```

An OPPL statement is decomposable in the following sections:

1. Variable definition (before SELECT)

2. Selection (between SELECT and BEGIN)

3. Actions (between BEGIN and END)

The grammar of an OPPL statement is shown at the Appendix A. Variables can have the following types: CLASS, OBJECTPROPERTY, DATAPROPERTY, INDIVIDUAL, and CONSTANT. Each variable type covers a possible category of entities in the OWL specification. An entity here is a named object or a constant. Therefore, an anonymous class description is not an acceptable substitution for any variable type, including CLASS. This limitation has important bearings on the possibility of building complete algorithms to execute an arbitrary query in OPPL.

The following script is an example of OPPL statement:

```
?c1:CLASS,  ?c2:CLASS
SELECT
   ?c1 subClassOf not ?c2
WHERE ?c1 != ?c2
BEGIN
   remove ?c1 subClassOf not ?c2,
   add ?c1 disjointWith ?c2
END;
```

2.3 Antipattern

We define antipatterns as patterns that appear obvious but are ineffective or far from optional in practice, representing worst practice about how to structure and build software [5]. We have also identified a set of patterns that are commonly used by domain experts in their DL-formalization and OWL implementations, and that normally result in unsatisfiable classes or modeling errors. Antipatterns come from a misuse and misunderstanding of DL expressions by ontology developers. We categorize them into three groups:

- Detectable Logical Antipatterns (DLAP). They represent errors that DL-reasoners and debugging tools normally detect.

- Cognitive Logical Antipatterns (CLAP). They represent possible modeling errors that may be due to a misunderstanding of the logical consequences of the used expression.

- Guidelines (G). They represent complex expressions used in an ontology component definition that are correct from the logical and cognitive points of view, but for which the ontology developer could have used other simpler alternatives or more accurate ones for encoding the same knowledge.

An antipattern is constructed by a detection pattern and some recommendations to repair the pattern. To illustrate the pattern, we use DL symbols that have the most compact form. We define formula here in this thesis as an expression that is formed by DL symbols and terms in the ontology.

The following table is the catalogue of antipatterns that has been collected in [5] (without their formalization of detection and recommendation):

Table 2.2: Catalogue of antipattern in [5]

No	Antipattern	Description	Category
1	AIO	AndIsOr	DLAP
2	OIL	OnlynessIsLoneliness	DLAP
3	UE	UniversalExistence	DLAP
4	UEWIP	UniversalExistenceWithInverseProperty	DLAP
5	EID	EquivalenceisDifference	DLAP
6	SOE	SynonymOrEquivalence	CLAP
7	DOC	DisjointnessOfComplement	G
8	DCC	Domain&CardinalityConstraints	G
9	DOC	GroupAxioms	G
10	DOC	MinIsZero	G

2.4 LintRoll

Lint in software means a program that is able to detect suspicious code that leads to unexpected behavior or bugs [12]. LintRoll is the name of plug-in built for Protégé that implements concept lint above. Therefore its main aim is to provide facilities to highlight pitfalls or antipatterns in one's ontologies. Lint here is a pattern that can be built from either an OPPL statement or Lint-JAR. A Lint-JAR is a lint implementation in Java Archive that we or some third party created. [†]

The connection OPPL with Lint detection is quite natural. Users can now declare what sort of axioms combinations are Lint and the OPPL engine will individuate such situations in the target ontologies, when present. Furthermore, users can now specify actions to undertake in case Lint is detected.

An OPPL Lint specification is made of the following components:

1. Name. A name must use English alphabet letters plus "–" and "_"

2. OPPL Script (see the OPPL Grammar on A)

3. Return Clause: "RETURN" followed by a valid variable name defined in the OPPL Script earlier.

4. Description: A natural language description of the Lint

Below there is an example of an OPPL Lint check already available in the Protégé 4 plug-in:

```
Transitive property lint;
?x:OBJECTPROPERTY, ?y:OBJECTPROPERTY
```

[†]The source of LintRoll is available at
http://www.cs.man.ac.uk/ iannonel/lintRoll/downloads.html

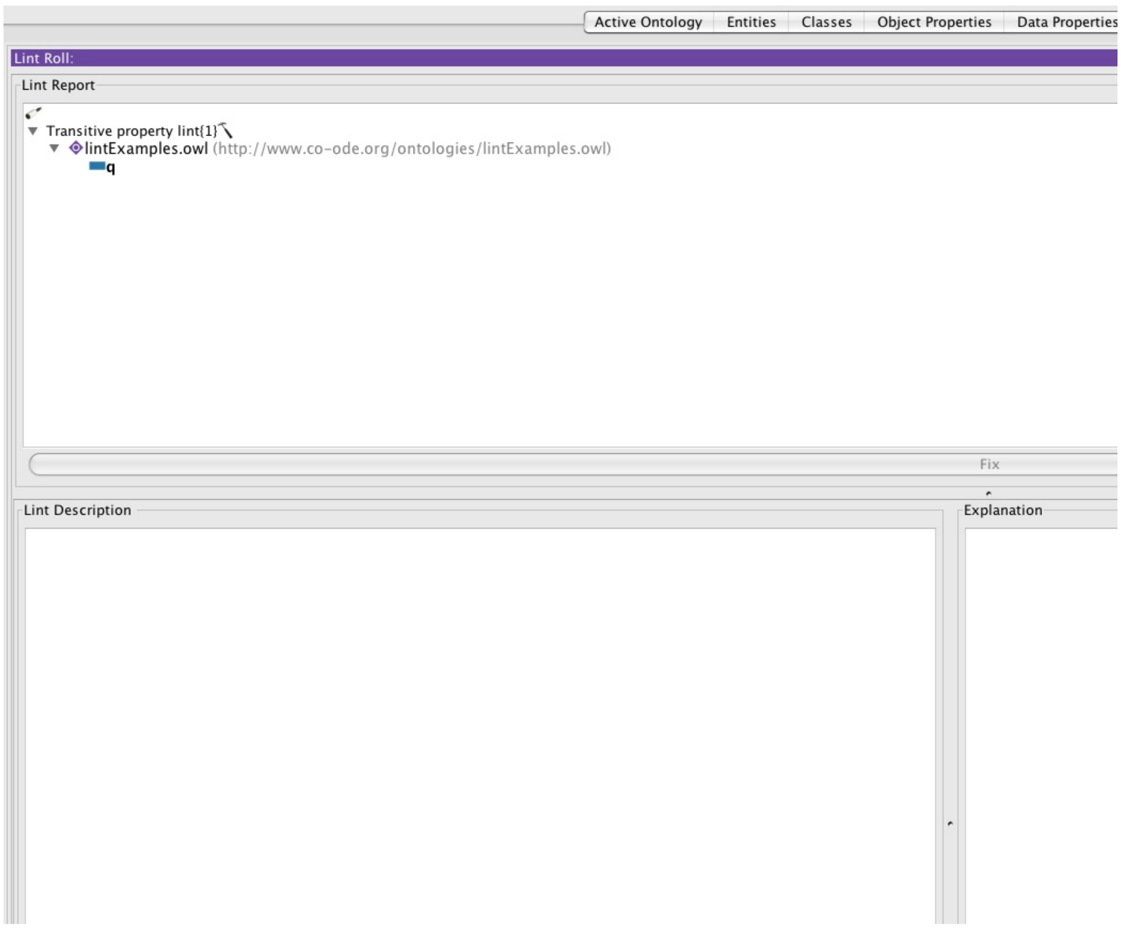


Figure 2.3: Screenshot of LintRoll

```
SELECT ?y subPropertyOf ?x, Transitive: ?y, Transitive: ?x WHERE ?x!= ?y
BEGIN
REMOVE Transitive: ?y
END;
RETURN ?y;
Lint that corrects the undesirable situation in which a transitive
object property is sub-property of another transitive object property
```

Users can add their own OPPL Lint checks. The OPPL Lint will be stored in the active ontology and will be kept there so that it will be available during its editing.

In order to create your own Lint-JAR, we have to implement Manchester OWL Lint Framework. There are four components that we must implement as follows:

1. Lint Interface

This is the main component in the framework. When developing a lint one must implement the following methods:

- *detected*() - In this method goes the actual code for verifying whether the situation the Lint instance has to detect occurs or not. Developers are supposed to return an implementation of LintReport containing the result of the detection process.
- *getDescription*() - This method should return a String containing a natural language description of the situation that the Lint detects
- $getName(), setName()$ - These are mere access methods for the name of the Lint used for display purposes

2. Lint Pattern Interface
A LintPattern is an interface that stands for a generic pattern matched across a set of OWLOntology instances. We need to implement method $matches()$. The implementation of this method that should an instance implementing the PatternReport interface containing the Set of $OWLObject$ which are matching this particular pattern.

3. Report (LintReport & PatternReport) interfaces
Both LintReport and PatternReport are interfaces containing the results of matching respectively a Lint or a LintPattern implementations. A developer needs to implement the following methods for both of them:

   - $getAffectedOntologies()$ - It returns the ontologies affected (in which there are OWLObject instances successfully matched) by the Lint (LintPattern).
   - $getAffectedOWLObjects()$ - Given an instance of OWLOntology this method should return the Set of OWLObject instances affected by (successfully matching) the Lint (LintPattern) that generated this implementation of this Report.
   - $isAffected(OWLOntology)$ should return true if the input OWLOntology instance is affected by (contains at least an OWLObject instance successfully matching) the Lint (LintPattern) that generated this report.
   - $getLint()(getLintPattern())$ - Should return the Lint (LintPattern) that generated this Report

4. PatternBasedLint Interface This interface abstracts over the kind of Lint implemented as a chain of LintPattern instances. The suggested semantics is that a PatternBasedLint is detected when all its LintPatterns successfully match a common subset of OWLObject instances. In addition to

those already described in Lint interface, a PatternBasedLint implementation should implements the getPatterns() method, which returns the Set of LintPattern instances this Lint is built upon.

Chapter 3

Work Objectives

This chapter discusses the work objectives of the thesis. It observes approaches to overcome problems that occur in the previous chapter. We present some hypotheses that lead us to assumptions and limitations.

From the discussion of chapter Introduction and State of the art, we define objective of this thesis as follows:

- to enrich the catalogue of antipattern by discovering new antipatterns. The first version of the catalogue as displayed on Table 6.3 contains only 10 antipatterns and has proposed a categorization of them. In this thesis we will perform a research to find new antipatterns.

  A finding technique that we use is a manual ontology debugging in which we expect to find some new antipatterns. The debugged ontology usually has some inconsistent classes after we classify with a reasoner. We apply the recommended action from the existing catalogue, and then we search the root cause of the rest of inconsistent classes.

  Once found, then we formalize it as detection of antipattern. The next step, we propose some actions to recover the problem and confirm them to the domain experts on what they really want. Later on, we formalize them as recommendation of antipattern.

  As we discussed, to debug ontology manually can be very hard even for experts. Discovering an antipattern in a complex ontology may be difficult. But this finding will help a lot to solve the same problem in different ontologies.

- to extend a formula representation of antipattern, so that a formula can have better representation of every single antipattern that is closer to its real data in an ontology.

We introduce new symbols $*$ (star) and $+$ (plus) in formulating antipatterns to complete DL-symbols on the current representation. These new symbols will explain every antipattern clearly. These are close to regular expressions, because these describe a search pattern too.

- to propose classification of antipattern according to its implementation.
  Aforementioned before, LintRoll is capable of detecting suspicious pattern that very often leads to unexpected behavior or bug. A pattern in LintRoll could be implemented in OPPL and Manchester OWL Lint Framework. Furthermore, as an antipattern itself is obviously a pattern as well, we want to classify an antipattern to either OPPL-based or any other implementation. We also want to make this new implementation be simpler than the one that Manchester OWL Lint Framework offers.

- to suggest transformation rule of axioms as additional feature.
  In order to help finding antipattern, we present an alternative way as a transformation rule to convert an axiom to another axiom with expectation that antipattern could be found. Indeed, this transformation will change the ontology, but as a trade of, we found an antipattern. The final decision must be returned to the ontology developer about whether his ontology modeling is correct or not.

- to build a plug-in as antipattern debugging tool.
  This debugging tool will solve time consumption issue and reduce complexity in ontology debugging. We expect that it will offer a clear direction to ontology developers.

- to compare the plug-in with another debugging tool.
  We use SWOOP as debugging tool comparator. We expect that this comparison will be able to identify pros and cons of the plug-in.

- to suggest a debugging strategy based on antipatterns.
  When debugging an ontology, ontology developers do some consecutive debugging actions until they find the ontology free of antipatterns. They select to focus on a certain antipattern, after that they go to another antipattern and so on. We will see if this is possible to perform a strategy of debugging.

Our hypotheses are:

- We are able to implement any antipatterns from the easiest way (OPPL) to the most difficult way (Lint), regardless of the degree of complexity.

- The developed plug-in can support ontology developers to debug an ontology effectively and efficiently.

Our assumptions are:

- The OPPL API works well, so it can fully support all functionality on the plug-in.
- Validation of a recommendation that is given by the plug-in must be done manually, so that participation of domain experts is needed, thus we assume that every taken action is valid.
- OWL API that we use is built-in of Protégé. Users must take into account the version of Protégé (at least its OWL API) is matched with what we use in our development.

Our limitations are:

- Research on antipattern is not stopped at this thesis. New antipatterns may be found in the future.
- We present a formula of antipattern on the plug-in but it will not affect the result of antipattern finding.
- Naming of an antipattern from a version of catalogue to the newer one could be changed. We cannot control if users want to change antipattern name from the original distribution of plug-in. This way, we will not talk about naming convention for antipattern.
- At times, an axiom transformation is needed. Since it is outside antipattern concept, we only implement one transformation rule.

Chapter 4

OPPL-Based AntiPatterns

Debugging ontology manually has yielded some discovery of new antipatterns. They will complete the old catalogue and the Table 4.1 shows our new catalogue of antipatterns.

OPPL and Lint-Jar are able to implement all antipatterns in the Table 4.1 but OPPL has some limitations. Following are some reasons why an antipattern could not be implemented by OPPL:

- Transitivity on subclass ($\sqsubseteq$) and equivalence ($\equiv$) relation
- Number of operands on the union ($\sqcup$) and intersection ($\sqcap$) could be arbitrary
- Special actions such as counting frequency of concepts used and removing a class

We cannot describe the transitivity on subclass and equivalence relation with OPPL. Some antipatterns use transitivity. For instance, when we are checking on the disjointness of two classes, we have to take into account the transitivity. There are two types of disjointness namely direct and indirect disjointness. The direct disjointness means there is a disjoint axiom that declares disjointness of two classes. For instance in HydrOntology, we will find the following axioms:

- $Disj(Aguas\_Continentales, Aguas\_Marinas)$;[*]
- $Disj(Aguas\_Subterráneas, Aguas\_Superficiales)$;
- $Disj(Afluente, Glaciar)$;

[*]In Protégé, it means $Aguas\_Continentales\ disjointWith\ Aguas\_Marinas$.

No	Antipattern	Description	Category
1	AIO	AndIsOr	DLAP
2	OIL	OnlynessIsLoneliness	DLAP
3	OILWI	OnlynessIsLonelinessWithInheritance	DLAP
4	UE	UniversalExistence	DLAP
5	UEWI	UniversalExistenceWithInheritance	DLAP
6	UEWPI	UniversalExistenceWithPropertyInheritance	DLAP
7	UEWIP	UniversalExistenceWithInverseProperty	DLAP
8	VOV	hasValueisOneValue	DLAP
9	EID	EquivalenceIsDifference	DLAP
10	EAD	EquivalenceAreDifferences	DLAP
11	SID	SubclassIsDifference	DLAP
12	SAD	SubclassAreDifferences	DLAP
13	MMCAR	MinimalMaximalCardinalityRestriction	DLAP
14	ECRWIP	Existential&CardinalityRestrictionWithInverseProperty	DLAP
15	SOSER	SumOfSomwithExactRestriction	DLAP
16	SOE	SynonymOrEquivalence	CLAP
17	SOS	SumOfSom	CLAP
18	DOC	DisjointnessOfComplement	G
19	UIE	UnionInEquivalency	G
20	ECR	Existential&CardinalityRestriction	G
21	SMALO	SomeMeansAtLeastOne	G
22	MIZ	MinIsZero	G
23	DOS	DistributivityOnSubclass	G
24	DCS	DisjointnessOfComplementonSubclass	G

Table 4.1: New catalogue of antipattern

The indirect disjointness between two classes is not explicitly mentioned by a disjoint axiom. It requires looking further on their superclass that possibly have a direct disjointness. On the other hand, we could formalize it as follows:

$$C_1 \sqsubseteq^+ C_3; C_2 \sqsubseteq^* C_4; Disj(C_3, C_4); \Longrightarrow Disj(C_1, C_2);^\dagger$$

We use pattern $C_1 \sqsubseteq^+ C_3$ to describe that there is at least one subclass relation involving either C_1 or C_3. For example, (1) $C_1 \sqsubseteq C_{11}; C_{11} \sqsubseteq C_3$; and (2) $C_1 \sqsubseteq C_{12}; C_{12} \equiv C_3$; belong to pattern $C_1 \sqsubseteq^+ C_3$. Meanwhile, we use pattern $C_2 \sqsubseteq^* C_4$ to express that subclass relation could be absent. It also could represent C_2 and C_4 are the same class. Examples for this pattern: (1) $C_2 \equiv C_4$;, (2) $C_2 \equiv C_{21}; C_{21} \equiv C_4$ and (3) $C_2 \equiv C_{21}; C_{21} \sqsubseteq C_4$.

By Venn Diagram Figure 4.1, it is obvious to prove the correctness of the disjointness between two classes if it is classified to the indirect disjointness.

$^\dagger Disj(C_3, C_4)$ is a direct disjointness

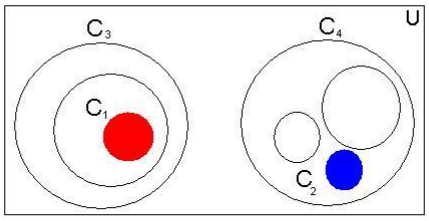


Figure 4.1: Indirect Disjointness of C_1 and C_2

Following is an example of the indirect disjointness:

- $Disj(Aguas\_Superficiales, Aguas\_Marinas)$, because:
  - $Aguas\_Superficiales \sqsubseteq Aguas\_Continentales$;[‡]
  - $Disj(Aguas\_Continentales, Aguas\_Marinas)$;

Sometimes, a class is unsatisfiable because of an axiom that yields the inconsistency originated from the ancestor of that class. This ancestor relationship also represents the transitivity on *subClassOf($\sqsubseteq$) and* equivalentTo($\equiv$) relation. For instance in Computer Science Ontology, following axioms cause $LecturerTaking4Courses$ be unsatisfiable:

- $LecturerTaking4Courses \sqsubseteq = 4\ takeCourse.\top$
- $TeachingFaculty \sqsubseteq \leq 3\ takeCourse.\top$
- $LecturerTaking4Courses \sqsubseteq Lecturer$
- $Lecturer \sqsubseteq TeachingFaculty$

OPPL could not provide a way in general to extract all axioms like on the example above. We also cannot express the operand of the union ($\sqcup$) and intersection ($\sqcap$) with OPPL, because the number of operands is arbitrary and could be any number. There is no way to create a general script that enables us to inquiry from an ontology.

The current version of OPPL only supports actions for adding and removing an axiom. How to remove a class is a thing that OPPL does not provide. On all antipatterns in category DLAP and CLAP, we will find usability of pattern transitivity, union, intersection and removing a class. Clearly according to some of the aforementioned facts, OPPL cannot support all antipatterns in category DLAP and CLAP.

On each OPPL-based antipattern below, we provide a script of OPPL to show how an OPPL-based antipattern implemented. An OPPL script consists of three

[‡]In Protégé, it means $Aguas\_Superficiales\ subClassOf\ Aguas\_Continentales$.

parts: variable declaration, query and action[§][14]. An antipattern can have more than one recommendation. A recommendation will yield the action part on a script. In this case, there is more than one action part of OPPL. Thus, we split the script into as many as number of actions that each script only has different on the action part.

4.1 Guideline DisjointnessOfComplement (DOC)

$$C_1 \equiv \ not\ C_2; \tag{4.1}$$

The ontology developer may want to say that C_1 and C_2 cannot share instances, instead of defining C_1 as the logical negation of C_2. Hence it could be more appropriate to state that C_1 and C_2 are disjoint. The following is an example of this antipattern in HydrOntology:

- $Laguna\_Salada \equiv \ not\ Aguas\_Dulces$
- $Salt\_Lagoon \equiv \ not\ Fresh\_Water$

We propose:

$$C_1 \cancel{\equiv}\cancel{not}\ C_2; \Rightarrow Disj(C_1, C_2); \tag{4.2}$$

After applying the above recommendation, corrections to be applied are as follows:

- $Disj(Laguna\_Salada, Aguas\_Dulces);$
- $Disj(Salt\_Lagoon, Fresh\_Water);$

The recommendation 4.2 yields a OPPL Script implementation for this antipattern which is displayed on the code 1.

4.2 Guideline SomeMeansAtLeastOne (SMALO)

$$C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq (\geq 1R.\top); \tag{4.3}$$

The cardinality restriction is superfluous, because if there is an existential restriction that means that the cardinality restriction using the same property is at least equal to 1. The ontology developer had created the axiom $C_1 \sqsubseteq (\geq 1R.\top)$ first, to say that C_1 should be defined by the R property. Next, he specialized his definition and forgot to remove the first restriction. In HydrOntology, this antipattern appears twice.

[§]Grammar of the OPPL script is available online at *http://oppl2.sourceforge.net/grammar.html.*

```
?c1:CLASS,  ?c2:CLASS
SELECT
   ?c1 equivalentTo not ?c2
WHERE ?c1 != ?c2
BEGIN
   remove ?c1 equivalentTo not ?c2,
   add ?c1 disjointWith ?c2
END;
```

Code 1: DOC OPPL Script

- $Estero \sqsubseteq \exists est\acute{a}_{p}roxima.Desembocadura; Estero \sqsubseteq (\geq 1\ est\acute{a}_{p}roxima.\top)$;
- $Rambla \sqsubseteq \exists es\_originado.Torrente; Rambla \sqsubseteq (\geq 1\ es\_originado.\top)$;

We recommend to remove the superfluous axiom.

$$C_1 \sqsubseteq \exists R.C_2; \cancel{C_1 \sqsubseteq (\geq 1\ R.\top);} \tag{4.4}$$

After applying the recommendation to the above examples, correction will be becoming as follows:

- $Estero \sqsubseteq \exists est\acute{a}_{p}roxima.Desembocadura$;
- $Rambla \sqsubseteq \exists es\_originado.Torrente$;

According to the recommendation above, Code 2 is a script of OPPL implementation for this antipattern.

```
?c1:CLASS, ?c2:CLASS,
?r:OBJECTPROPERTY
Select
   ?c1 subClassOf ?r some ?c2,
   ?c1 subClassOf ?r min 1 Thing
Where ?c1 != ?c2
begin
   remove  ?c1 subClassOf ?r min 1 Thing
end;
```

Code 2: SMALO OPPL Script

4.3 Guideline MinIsZero (MIZ)

$$C_1 \sqsubseteq (\geq 0 R.\top); \tag{4.5}$$

The ontology developer wants to say that C_1 can be the domain of the R property. This restriction has no impact on the logical model being defined and can be removed. This antipattern appeared once in the HydrOntology debugging process.

- $Laguna\_Salada \sqsubseteq (\geq 0\ es\_alimentada.\top)$;

Hence, we propose to remove the axiom.

$$\cancel{C_1 \sqsubseteq (\geq 0 R.\top);} \tag{4.6}$$

According to the recommendation above, a script of OPPL implementation for this antipattern will be like Code 3.

```
?c1:CLASS,
?r:OBJECTPROPERTY
Select
   ?c1 subClassOf ?r min 0 Thing
Where ?c1 != Thing
begin
   remove ?c1 subClassOf ?r min 0 Thing
end;
```

Code 3: MIZ OPPL Script

4.4 Guideline DisjointnessofComplementonSubclass (DCS)

$$C_1 \sqsubseteq \ not\ C_2; \tag{4.7}$$

The ontology developer may want to say that C_1 and C_2 cannot share instances, instead of defining C_1 as subclass of the logical negation of C_2. Conversion the current axiom into a disjoint axiom is absolutely allowed since it does not change the semantic. Hence it could be more appropriate to state that C_1 and C_2 are disjoint. The following are axioms discovered as this antipattern in HydrOntology:

- $Laguna\_Salada \sqsubseteq \ not\ Aguas\_Dulces$

- $Agua\_Marinas \sqsubseteq$ *not* $Aguas\_Dulces$
- $Albufera \sqsubseteq$ *not* $Aguas\_Dulces$

We propose to change the axiom into a disjointness axiom because of the equivalency.

$$C_1 \cancel{\sqsubseteq not}\ C_2; \Rightarrow Disj(C_1, C_2); \tag{4.8}$$

The examples above after correction will changed to:

- $Disj(Laguna\_Salada, Aguas\_Dulces)$;
- $Disj(Agua\_Marinas, Aguas\_Dulces)$;
- $Disj(Albufera, Aguas\_Dulces)$;

A OPPL script to implement this antipattern should be like code 4.

```
?c1:CLASS,  ?c2:CLASS
SELECT
   ?c1 subClassOf not ?c2
WHERE ?c1 != ?c2
BEGIN
   remove ?c1 subClassOf not ?c2,
   add ?c1 disjointWith ?c2
END;
```

Code 4: DCS OPPL Script

Chapter 5

Lint-Based AntiPatterns

Lint-Based antiPatterns are a collection of antipatterns that could not be implemented by OPPL. Since, we have Java implementation inside of Lint, Lint is more powerful than OPPL. It can handle a complex antipattern that requires a specific query (detection) and action. Nevertheless, Lint is not easier to implement compared to OPPL.

The idea of Lint comes from the implementation of the Lint-Jar in LintRoll plug-in on Protégé. We simplify the use of the Lint-Jar and come up with an implementation which covers every function needed to support an antipattern. We also think about reusability. Probably, two antipatterns have the same action part. Thus, one implementation could be used for two antipatterns.

Some antipatterns may contain a disjoint axiom. Lint is able to detect either direct or indirect disjointness between two classes. Furthermore, Lint also can detect disjointness over two unions as long as all components in each union are a class. The Figure 5.1 illustrates the disjointness over two unions. C_1 and C_2 are not necessarily disjoint, neither are C_3, C_4 and C_5.

If we find an axiom $Disj(C_1, C_2)$, notice that this does not mean that the ontology developer has explicitly expressed that C_1 and C_1 are disjoint, but that these two concepts are determined as disjoint from each other by a reasoner

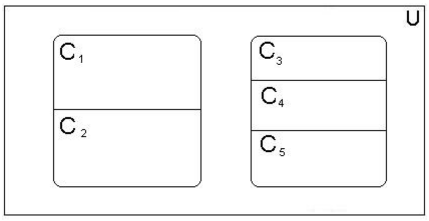


Figure 5.1: Indirect Disjointness over two unions

(Facts++, Pellet, etc). We use this notation as a shorthand for $C_1 \sqcap C_2 \sqsubseteq \bot$

In order to present how Lint implements an antipattern, we provide a pseudocode [6] for helping us to understand how an antipattern is implemented. Pseudocode is an artificial and informal language that helps programmers develop algorithms. Pseudocode is a "text-based" detail (algorithmic) design tool. Each antipattern consists of two parts, namely detection and action. Detection and action have their own pseudocode. Since an antipattern may have several detection pattern and different actions, the pseudocode for the antipattern could be more than two. A detection process will return pairs of lists of axioms and lists of parameters for action process, while an action process will return lists of actions where the user will choose the most appropriate one.

5.1 Detectable Logical AntiPatterns (DLAP)

5.1.1 AntiPattern AndIsOr (AIO)

$$C_1 \sqsubseteq \exists R.(C_{21} \sqcap ... \sqcap C_{2n}); Disj(C_{2i}, C_{2j}); \tag{5.1}$$

$$C_1 \sqsubseteq (C_{21} \sqcap ... \sqcap C_{2n}); Disj(C_{2i}, C_{2j}); \tag{5.2}$$

This is a common modeling error that appears due to the fact that in common linguistic usage, "and" and "or" do not correspond consistently to logical conjunction and disjunction respectively [18]. For example, "I like cake with milk and chocolate" is ambiguous. Does the cake contain?

- Some chocolate plus some milk? $Cake \sqsubseteq \exists contain.Chocolate \sqcap \exists contain.Milk$
- Chocolate-flavoured milk? $Cake \sqsubseteq \exists contain.(Chocolate \sqcap Milk)$
- Some chocolate or some milk? $Cake \sqsubseteq \exists contain.(Chocolate \sqcup Milk)$

Notice that the second version of the AIO antipattern 5.1 is contained in the first one with an anonymous class. In the original version of HydrOntology this antipattern appeared twice. The following is one instance of this antipattern :[*]

- $Caño \sqsubseteq \exists comunica.(Albufera \sqcap Mar \sqcap Marisma);$

In order to solve this antipattern we propose replacing the logical conjunction by the logical disjunction, or by the conjunction of two existential restrictions.

$$\left.\begin{array}{r} \cancel{C_1 \sqsubseteq \exists R.(C_{21} \sqcap ... \sqcap C_{2n});} \\ Disj(C_{2i}, C_{2j}); \end{array}\right\} \Rightarrow \quad C_1 \sqsubseteq \exists R.(C_{21} \sqcup ... \sqcup C_{2n}); \tag{5.3}$$

$$\Rightarrow \quad C_1 \sqsubseteq (\exists R.C_{21}) \sqcap ... \sqcap (\exists R.C_{2n}); \tag{5.4}$$

[*]For better readability, we do not specify in these examples that the used classes are disjoint

$$\cancel{C_1 \sqsubseteq C_{21} \sqcap ... \sqcap C_{2n}}; Disj(C_{2i}, C_{2j}); \quad \Rightarrow \quad C_1 \sqsubseteq C_{21} \sqcup ... \sqcup C_{2n}; \tag{5.5}$$

After applying all recommendations above, we have possible corrections for the example of this antipattern as follows:

- Recommendation 5.3:
  $Caño \sqsubseteq \exists comunica.(Albufera \sqcup Mar \sqcup Marisma)$;

- Recommendation 5.4:
  $Caño \sqsubseteq (\exists comunica.Albufera) \sqcap (\exists comunica.Mar) \sqcap (\exists comunica.Marisma)$

We provide several pseudocodes below that illustrate the detection (Code 5 and Code 6) and action process (Code 7, Code 8 and Code 9) to show how to implement this antipattern.

Code 5 Pseudocode for AIO (5.1)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom$ in $subClassAxioms$
    let $(subClass \sqsubseteq superClass) = axiom$ *
    if $superClass$ is $\exists$ restriction
        let $\exists R.intersection = superClass$
        if $\exists C_i, C_j$ in operands of intersection where $Disj(C_i, C_j)$
            $disjointAxioms :=$ get all axioms performing
                                the disjointness of $C_i$ and $C_j$
            add $([axiom] + disjointAxioms, [subClass, R, operands, axiom])$
                into $results$ †
return $results$
```

*We use notation 'let ... = ...' [21] as a pattern matching of both side of '='. Example : 'let $(a \sqsubseteq b) = (Caño \sqsubseteq \exists comunica.(Albufera \sqcap Mar \sqcap Marisma))$' yields $a := Caño$ and $b := \exists comunica.(Albufera \sqcap Mar \sqcap Marisma)$

†Notation $[x] + [y]$ defines a concatenation between two lists or arrays and the result is $[x, y]$. Meanwhile, notation 'add x into L' defines that the element x is appended into the list or array L.

Code 6 Pseudocode for AIO (5.2)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axioms in the ontology
for each $axiom$ in $subClassAxioms$
    let $(subClass \sqsubseteq superClass) = axiom$
    if $superClass$ is an intersection
      if $\exists C_i, C_j$ in operands of intersection where $Disj(C_i, C_j)$
        $disjointAxioms$ := get all axioms performing
                                the disjointness of $C_i$ and $C_j$
        add $([axiom] + disjointAxioms, [subClass, operands, axiom])$
            into $results$
return $results$
```

Code 7 Pseudocode for AIO (5.3)

```
Load var $[subClass, R, operands, axiom]$ from detection process
$removedAction$ := create an action to remove $axiom$
$newAxiom := subClass \sqsubseteq \exists R.UnionOf(operands)$
$addedAction$ := create an action to add $newAxiom$
return $[removedAction,\ addedAction]$
```

Code 8 Pseudocode for AIO (5.4)

```
Load var $[subClass, R, operands, axiom]$ from detection process
$removedAction$ := create an action to remove $axiom$
$addedActionList$ is a list
for each operand in operands
    $newAxiom := subClass \sqsubseteq \exists R.operand$
    $addedAction$ := create an action to add $newAxiom$
    add $addedAction$ into $addedActionList$
return $[removedAction]$+$addedActionList$
```

Code 9 Pseudocode for AIO (5.5)

```
Load var $[subClass, operands, axiom]$ from detection process
$removedAction$ := create an action to remove $axiom$
$newAxiom := subClass \sqsubseteq UnionOf(operands)$
$addedAction$ := create an action to add $newAxiom$
return $[removedAction,\ addedAction]$
```

5.1.2 AntiPattern OnlynessIsLoneliness (OIL)

$$C_1 \sqsubseteq \forall R.C_2; C_1 \sqsubseteq \forall R.C_3; Disj(C_2, C_3); \tag{5.6}$$

It is necessary to be detectable, property R must have at least a value, normally specified as existential restrictions, (minimum) or exact cardinality restriction for that class with a positive number on the cardinality.

The ontology developer created a universal restriction to say that C_1 instances can only be linked with property R to C_2 instances. Next, a new universal restriction is added saying that C_1 instances can only be linked with R to C_3 instances, with C_2 and C_3 disjoint. In general, this is because the ontology developer forgot the previous axiom in the same class or in any of the parent classes. The following is one of the two examples of this antipattern in HydrOntology:[†]

- $Aguas\_de\_Transición \sqsubseteq \forall está\_próxima.Aguas\_Marinas;$
  $Aguas\_de\_Transición \sqsubseteq \forall está\_próxima.Desembocadura;$

If it makes sense, we propose to the ontology developer to transform the two universal restrictions into only one that refers to the logical disjunction of C_2 and C_3.

$$\cancel{C_1 \sqsubseteq \forall R.C_2; C_1 \sqsubseteq \forall R.C_3;} Disj(C_2, C_3); \quad \Rightarrow \quad C_1 \sqsubseteq \forall R.(C_2 \sqcup C_3); \tag{5.7}$$

According to the recommendation and to apply it on the example, we will get new axiom :

- $Aguas\_de\_Transición \sqsubseteq \forall está\_próxima.(Aguas\_Marinas \sqcap Desembocadura);$

In order to show how to implement this antipattern, we provide a pseudocode of detection (Code 10) and action process (Code 11) below.

[†]In the example, for readability reason, we do not show a related axiom with property R must have at least a value and disjointness.

Code 10 Pseudocode for OIL (5.6)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
  let $(C_1 \sqsubseteq superClass) = axiom$
  if $superClass$ is not $\forall$ restriction
    continue to the next loop
  let $\forall R.C_2 = superClass$
  if neither $C_2$ is a class nor UnionOf
    continue to the next loop
  $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_1$
  for each $axiom2$ in $subClassAxioms2$
    let $(C_1 \sqsubseteq superClass) = axiom2$
    if $superClass$ is not $\forall$ restriction
      continue to the next loop [*]
    let $\forall R.C_3 = superClass$
    if $C_3 == C_2$ or neither $C_3$ is a class nor UnionOf [†]
      continue to the next loop
    if not $Disj(C_2, C_3)$
      continue to the next loop
    if $C_1$ or its ancestor have $\exists R.C$ or $= mR.C$ or $\geq nR.C$ where $m, n > 0$
      $disjointAxioms$ := get all axioms performing
                          the disjointness of $C_2$ and $C_3$
      $SomeExactMinAxioms$ :=get all axioms performing
                                 the clause on 'if statement' above
      $axiomList$ :=[$axiom1, axiom2$]+
                    $disjointAxioms + SomeExactMinAxioms$
      $paramList$ := $[axiom1, axiom2, C_1, R, C_2, C_3]$
      add pair $(axiomList, paramList)$ into $results$
return $results$
```

[*]It means that the execution will continue to the next loop on the current loop

[†]$C_3 == C_2$ represents that C_3 and C_2 are the same symbol (class, anonymous class, property, etc), and it does not represent that C_3 and C_2 are semantically equivalent. C_3 is a class meaning that C_3 is a proper class and not anonymous.

Code 11 Pseudocode for OIL (5.7)

```
Load var $[axiom1, axiom2, C_1, R, C_2, C_3]$ from detection process
$removedAction1$ := create an action to remove $axiom1$
$removedAction2$ := create an action to remove $axiom2$
$newAxiom := C_1 \sqsubseteq \forall R.(C_2 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return $[removedAction1, removedAction2, addedAction]$
```

5.1.3 AntiPattern OnlynessIsLonelinessWithInheritance (OILWI)

$$C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; C_2 \sqsubseteq \forall R.C_4; Disj(C_3, C_4); \tag{5.8}$$

Like the antipattern OIL, it is necessary to be detectable, property R must have at least a value, normally specified as existential restrictions, (minimum) or exact cardinality restriction for that class with a positive number on the cardinality.

The ontology developer has added a universal restriction for class C_1 without remembering that he had already defined another universal restriction with the same property in a parent class. This incoherence comes from the fact that the subclass inherits from its parent all its constraints. This antipattern is a specialization of OIL. This antipattern appeared twice in HydrOntology debugging process.

- $Ibon \sqsubseteq Charca; Ibon \sqsubseteq \forall es\_originado.(Glaciar \sqcup Masa\_de\_Hielo);$
  $Charca \sqsubseteq \forall es\_originado.(Arroyo \sqcup Manantial \sqcup Rio);$

- $Albufera \sqsubseteq Laguna; Laguna \sqsubseteq Aguas\_Quietas\_Naturales;$
  $Albufera \sqsubseteq \forall es\_alimentada.Aguas\_Marinas;$
  $Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.Aguas\_Corrientes\_Naturales;$

To solve this antipattern, the ontology developer should follow the OIL recommendation apply on the parent class C_2. Because a child class inherit all the axioms of its parent, all the axioms of the parents should apply on the child too.

$$\left.\begin{array}{r} C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; \\ \cancel{C_2 \sqsubseteq \forall R.C_4;} Disj(C_3, C_4); \end{array}\right\} \Rightarrow \quad C_2 \sqsubseteq \forall R.(C_3 \sqcup C_4); \tag{5.9}$$

From the examples of this antipattern, the recommendation 5.9 yields the following axioms:

- $Ibon \sqsubseteq Charca; Ibon \sqsubseteq \forall es\_originado.(Glaciar \sqcup Masa\_de\_Hielo);$ $Charca \sqsubseteq \forall es\_originado.(Glaciar \sqcup Masa\_de\_Hielo \sqcup Arroyo \sqcup Manantial \sqcup Rio);$

- $Albufera \sqsubseteq Laguna; Laguna \sqsubseteq Aguas\_Quietas\_Naturales;$ $Albufera \sqsubseteq \forall es\_alimentada.Aguas\_Marinas;$ $Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.(Aguas\_Marinas \sqcup Aguas\_Corrientes\_Naturales);$

The code 12 is a pseudocode for detection process of this antipattern, and the code 13 is a pseudocode for the action one.

Code 12 Pseudocode for OILWI (5.8)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
  let $(C_1 \sqsubseteq superClass) = axiom$
  if $superClass$ is not $\forall$ restriction
    continue to the next loop
  let $\forall R.C_3 = superClass$
  if neither $C_3$ is a class nor UnionOf
    continue to the next loop
  for each $axiom2$ in $subClassAxioms$, but different with $axiom1$
    let $(C_2 \sqsubseteq superClass) = axiom2$
    if $superClass$ is not $\forall$ restriction
      continue to the next loop
    let $\forall R.C_4 = superClass$
    if $C_1 == C_2$ and neither $C_4$ is a class nor UnionOf
      continue to the next loop
    if not $Disj(C_3, C_4)$
      continue to the next loop
    if $C_1 \sqsubseteq^+ C_2$ is not hold
      continue to the next loop
    if $C_1$ or its ancestor have $\exists R.C$ or $= mR.C$ or $\geq nR.C$ where $m, n > 0$
      $SomeExactMinAxioms$ :=get all axioms performing
                                  the clause on 'if statement' above
      $disjointAxioms$ := get all axioms performing
                              the disjointness of $C_3$ and $C_4$

      ...(continue to the next page)
```

```
        ...
        $AncestorAxioms$ := get all axioms performing $C_1 \sqsubseteq^+ C_2$
        $axiomList$ :=$AncestorAxioms + [axiom1, axiom2] + disjointAxioms+$
                        $SomeExactMinAxioms$
        $paramList := [C_2, C_3, C_4, R, axiom2]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 13 Pseudocode for OILWI (5.9)

```
Load var $[C_2, C_3, C_4, R, axiom2]$ from detection process
$removedAction$ := create an action to remove $axiom2$
$newAxiom := C_2 \sqsubseteq \forall R.C_3 \sqcup C_4$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction$, $addedAction$]
```

5.1.4 AntiPattern UniversalExistence (UE)

$$C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq \forall R.C_3; Disj(C_2, C_3); \tag{5.10}$$

The ontology developer adds an existential/universal restriction to a class whilst there was already an inconsistency-leading universal/existential restriction in the same class or in a parent class, respectively. The following is one of three examples of this antipattern in HydrOntology:

- $Gola \sqsubseteq Canal\_Aguas\_Marinas; Gola \sqsubseteq \exists comunica.Ria;$
  $Canal\_Aguas\_Marinas \sqsubseteq \forall comunica.Aguas\_Marinas;$

These antipatterns are difficult to debug because ontology developers sometimes do not distinguish clearly between existential and universal restrictions. Our proposal is aimed at resolving the unsatisfiability of a class, but as usual it should be clearly analyzed by the ontology developer.

$$C_1 \sqsubseteq \exists R.C_2; \cancel{C_1 \sqsubseteq \forall R.C_3;} Disj(C2, C3); \quad \Rightarrow \quad C_1 \sqsubseteq \forall R.(C_2 \sqcup C_3); \tag{5.11}$$

Thanks to this recommendation, the correction of the example would be:

- $Gola \sqsubseteq \exists comunica.Ria; Gola \sqsubseteq \forall comunica.(Aguas\_Marinas \sqcup Ria)$;

We provide several pseudocodes below that illustrate the detection (Code 14) and action process (Code 15) to show how to implement this antipattern.

Code 14 Pseudocode for UE (5.10)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom$
    if $superClass$ is not $\exists$ restriction
        continue to the next loop
    let $\forall R.C_2 = superClass$
    if neither $C_2$ is a class nor UnionOf
        continue to the next loop
    $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_1$
    for each $axiom2$ in $subClassAxioms2$
        let $(C_1 \sqsubseteq superClass) = axiom2$
        if $superClass$ is not $\forall$ restriction
            continue to the next loop
        let $\forall R.C_3 = superClass$
        if $C_3 == C_2$ or neither $C_3$ is a class nor UnionOf †
            continue to the next loop
        if $Disj(C_2, C_3)$
            $disjointAxioms$ := get all axioms performing
                                the disjointness of $C_2$ and $C_3$
            $axiomList := [axiom1, axiom2] + disjointAxioms$
            $paramList := [C_1, C_2, C_3, R, axiom2]$
            add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 15 Pseudocode for UE (5.11)

```
Load var $[C_1, C_2, C_3, R, axiom2]$ from detection process
$removedAction$ := create an action to remove $axiom2$
$newAxiom := C_2 \sqsubseteq \forall R.(C_2 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction$, $addedAction$]
```

5.1.5 AntiPattern UniversalExistenceWithInheritance (UEWI)

$$
\begin{aligned}
&C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \exists R.C_3; \quad C_2 \sqsubseteq \forall R.C_4; Disj(C_3, C_4);\\
&C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; \quad C_2 \sqsubseteq \exists R.C_4; Disj(C_3, C_4);
\end{aligned}
\tag{5.12}
$$

The ontology developer has added an axiom in a subclass without remembering that he has already defined the parent class with the R property. This incoherence comes from the fact that subclass inherits from its parent all its constraints. This AntiPattern is a specialization of UE.

This antipattern appeared three times in HydrOntology debugging process.

- $Gola \sqsubseteq Canal\_Aguas\_Marinas;$
  $Gola \sqsubseteq \exists comunica.R\acute{\imath}a;$
  $Canal\_Aguas\_Marinas \sqsubseteq \forall comunica.Aguas\_Marinas;$

- $Laguna\_Salada \sqsubseteq Laguna;$
  $Laguna \sqsubseteq Aguas\_Quietas\_Naturales;$
  $Laguna\_Salada \sqsubseteq \exists es\_alimentada.Aguas\_Marinas;$
  $Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.Aguas\_Corrientes\_Naturales;$

- $Ib\acute{o}n \sqsubseteq Charca$
  $Ib\acute{o}n \sqsubseteq \forall es\_originado.(Glaciar \sqcup Masa\_de\_Hielo);$
  $Charca \sqsubseteq \exists es\_originado.(Arroyo \sqcup Manantial \sqcup R\acute{\imath}o);$

These two antipatterns are very difficult to debug and depend of the ontology developer point of view. We propose a logical correction of these two antipatterns in order to obtain a coherent taxonomy. But you need to discuss with the ontology developer to be sure that this is what he wants to say. Maybe a more simple solution is possible depending of the relation between C_3 and C_4.

$$
\begin{aligned}
&\left.\begin{array}{r} C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \exists R.C_3;\\ \cancel{C_2 \sqsubseteq \forall R.C_4;} Disj(C_3, C_4); \end{array}\right\} \Rightarrow \quad C_2 \sqsubseteq \forall R.(C_3 \sqcup C_4);\\
&\left.\begin{array}{r} C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3;\\ \cancel{C_2 \sqsubseteq \exists R.C_4;} Disj(C_3, C_4); \end{array}\right\} \Rightarrow \quad C_2 \sqsubseteq \exists R.(C_3 \sqcup C_4);
\end{aligned}
\tag{5.13}
$$

Thanks for this recommendation, the correction of examples would be:

- $Gola \sqsubseteq Canal\_Aguas\_Marinas;$
  $Gola \sqsubseteq \exists comunica.R\acute{\imath}a;$
  $Canal\_Aguas\_Marinas \sqsubseteq \forall comunica.(R\acute{\imath}a \sqcup Aguas\_Marinas);$

- $Laguna\_Salada \sqsubseteq Laguna$;
  $Laguna \sqsubseteq Aguas\_Quietas\_Naturales$;
  $Laguna\_Salada \sqsubseteq \exists es\_alimentada.Aguas\_Marinas$;
  $Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.(Aguas\_Marinas \sqcup Aguas\_Corrientes\_Naturales)$;

- $Ibón \sqsubseteq Charca$
  $Ibón \sqsubseteq \forall es\_originado.(Glaciar \sqcup Masa\_de\_Hielo)$;
  $Charca \sqsubseteq \exists es\_originado.(Glaciar \sqcup Masa\_de\_Hielo \sqcup Arroyo \sqcup Manantial \sqcup Río)$;

We show only a pseudocode of the first type of detection. The second type is simmetric with the first one by exchanging ∃ and ∀ on the detection and action process. The detection process of this antipattern has the pseudocode Code 16. The first type of this antipattern has similar pseudocode with OILWI (see 5.9).

Code 16 Pseudocode for UEWI (5.12)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
  let $(C_1 \sqsubseteq superClass) = axiom$
  if $superClass$ is not ∃ restriction
    continue to the next loop
  let $\exists R.C_3 = superClass$
  if neither $C_3$ is a class nor UnionOf
    continue to the next loop
  for each $axiom2$ in $subClassAxioms$, but differentWith $axiom1$
    let $(C_2 \sqsubseteq superClass) = axiom2$
    if $superClass$ is not ∀ restriction
      continue to the next loop
    let $\forall R.C_4 = superClass$
    if $C_1 == C_2$ and neither $C_4$ is a class nor UnionOf
      continue to the next loop
    if not $Disj(C_3, C_4)$
      continue to the next loop

    ...(continue to the next page)
  ...
  if $C_1 \sqsubseteq^+ C_2$ holds
    $disjointAxioms$ := get all axioms performing
                          the disjointness of $C_3$ and $C_4$
    $AncestorAxioms$ := get all axioms performing $C_1 \sqsubseteq^+ C_2$
    $axiomList := AncestorAxioms + [axiom1, axiom2] + disjointAxioms$
    $paramList := [C_2, C_3, C_4, R, axiom2]$
    add pair $(axiomList, paramList)$ into $results$
return $results$
```

5.1.6 AntiPattern UniversalExistenceWithPropertyInheritance (UEWPI)

$$\begin{aligned} R_1 \sqsubseteq R_2{}^{\ddagger}; C_1 \sqsubseteq \exists R_1.C_2; C_1 \sqsubseteq \forall R_2.C_3; Disj(C_2, C_3);\\ R_1 \sqsubseteq R_2; C_1 \sqsubseteq \forall R_1.C_2; C_1 \sqsubseteq \exists R_2.C_3; Disj(C_2, C_3); \end{aligned} \tag{5.14}$$

The ontology developer misunderstands the sub-property relation between properties, thinking that it is similar to a part-of relation. This antipattern is a specialization of UE because $C_1 \sqsubseteq \exists R_1.C_2; R_1 \sqsubseteq R_2 \models C_1 \sqsubseteq \exists R_2.C_2$.
This antipattern appeared 1 time in HydrOntology debugging process.

- $se\_extrae \sqsubseteq es\_alimentada$;
  $Fuente\_Artificiale \sqsubseteq \exists se\_extrae.Acuífero$;
  $Fuente\_Artificiale \sqsubseteq \forall es\_alimentada.Tubería$;

Like for UE there may exist several recommendations for this antipattern. In our experiment, we first propose to the ontology developer the UE recommendation (see equation 5.15). But the ontology developer does not validate this solution. After some discussion ans studies, we have realized that he misunderstood the subclass-of relation between property. Thus, the solution was to remove the sub property relation between R_1 and R_2.

$$\begin{aligned} \left.\begin{array}{rl} R_1 \sqsubseteq R_2; & C_1 \sqsubseteq \exists R_1.C_2;\\ \cancel{C_1 \sqsubseteq \forall R_2.C_3}; & Disj(C_2, C_3); \end{array}\right\} &\Rightarrow C_1 \sqsubseteq \forall R_2.(C_2 \sqcup C_3);\\ \left.\begin{array}{rl} R_1 \sqsubseteq R_2; & C_1 \sqsubseteq \forall R_1.C_2;\\ \cancel{C_1 \sqsubseteq \exists R_2.C_3}; & Disj(C_2, C_3); \end{array}\right\} &\Rightarrow C_1 \sqsubseteq \exists R_2.(C_2 \sqcup C_3); \end{aligned} \tag{5.15}$$

[‡] R_1 is a sub property of R_2 or we use an axiom $R_1 subPropertOf R_2$ in Protégé.

$$\begin{aligned} &\cancel{R_1 \sqsubseteq R_2}; C_1 \sqsubseteq \exists R_1.C_2; C_1 \sqsubseteq \forall R_2.C_3; Disj(C_2, C_3); \\ &\cancel{R_1 \sqsubseteq R_2}; C_1 \sqsubseteq \forall R_1.C_2; C_1 \sqsubseteq \exists R_2.C_3; Disj(C_2, C_3); \end{aligned} \tag{5.16}$$

Because of the second recommendation, we remove sub property axiom on the examples. The correction of examples after applying the first recommendation would be:

- $se\_extrae \sqsubseteq es\_alimentada$;
  $Fuente\_Artificiale \sqsubseteq \exists se\_extrae.Acuí fero$;
  $Fuente\_Artificiale \sqsubseteq \forall es\_alimentada.(Acuí fero \sqcup Tubería)$;

An implementation of this antipattern is more complex than antipattern UEWI since it involves a sub property axiom. We provide an example implementation for the first type of this antipattern (Code 17, Code 18 and Code 19) . The second type is symmetric of the first one by exchanging $\exists$ and $\forall$.

Code 17 Pseudocode for UEWPI (5.14)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
  let $(C_1 \sqsubseteq superClass) = axiom$
  if $superClass$ is not $\exists$ restriction
    continue to the next loop
  let $\exists R_1.C_2 = superClass$
  if neither $C_2$ is a class nor UnionOf
    continue to the next loop
  $superPropertiesR1$ := get all super properties of $R_1$
  if $|superPropertiesR1| == 0$ [*]
    continue to the next loop
  $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_1$

  ...(continue to the next page)
```

[*] $|L|$ means size of list or set L.

```
 for each $axiom2$ in $subClassAxioms2$
     let $(C_1 \sqsubseteq superClass) = axiom2$
     if $superClass$ is not $\forall$ restriction
       continue to the next loop
     let $\forall R_2.C_3 = superClass$
     if $R_2 \notin superPropertiesR1$
       continue to the next loop
     if $C_3 == C_2$ or neither $C_3$ is a class nor UnionOf
       continue to the next loop
     if $Disj(C_2, C_3)$
       $subPropertyAxiom$ := get the axiom performing $R_1 \sqsubseteq R_2$
       $disjointAxioms$ := get all axioms performing
                                   the disjointness of $C_2$ and $C_3$
       $axiomList$ := $[axiom1, axiom2, subPropertyAxiom] + disjointAxioms$
       $paramList$ := $[C_1, C_2, C_3, R_2, axiom2, subPropertyAxiom]$
       add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 18 Pseudocode for UEWPI (5.15)

```
Load var $[C_1, C_2, C_3, R_2, axiom2, subPropertyAxiom]$ from detection process
$removedAction$ := create an action to remove $axiom2$
$newAxiom$ := $C_2 \sqsubseteq \forall R_2.(C_2 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return $[removedAction, addedAction]$
```

Code 19 Pseudocode for UEWPI (5.16)

```
Load var $[C_1, C_2, C_3, R_2, axiom2, subPropertyAxiom]$ from detection process
$removedAction$ := create an action to remove $subPropertyAxiom$
return $[removedAction]$
```

5.1.7 AntiPattern UniversalExistenceWithInverseProperty (UEWIP)

$$\begin{gathered} C_2 \sqsubseteq \exists R^-.{C_{1a}}^{\S}; C_{1b} \sqsubseteq \forall R.C_3; Disj(C_2, C_3); C_{1a} \sqsubseteq^* C_{1b}; \\ C_2 \sqsubseteq \forall R^-.C_{1a}; C_{1b} \sqsubseteq \exists R.C_3; Disj(C_2, C_3); C_{1a} \sqsubseteq^* C_{1b}; \end{gathered} \tag{5.17}$$

The ontology developer added restrictions about C_2, C_{1a} and C_{1b}[¶] using a property and its inverse. This antipattern is a specialization of UEWI and SOS because:

- $C_2 \sqsubseteq \exists R^-.C_{1a}; \models C_{1.1} \sqsubseteq \exists R.C_2; C_{1.1} \sqsubseteq C_{1a}$; which is a UEWI antipattern. See equation 5.12.
- $C_2 \sqsubseteq \forall R^-.C_{1a}$; imply that it may exist a class $C_{1.1} \sqsubseteq C_{1a}$; that can be defined as $C_{1.1} \sqsubseteq \exists R.C_2$; which is a specialization of SOS antipattern. See equation 5.40.

This antipattern appeared twice in HydrOntology debugging process.

- $Mar \sqsubseteq \exists alimenta.Albufera;$
  $Albufera \sqsubseteq Laguna;$
  $Laguna \sqsubseteq Aguas\_Quietas\_Naturales;$
  $alimenta \equiv es\_alimentada^-;$ [‖]
  $Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.Aguas\_Corrientes\_Naturales$
- $R\acute{i}o \sqsubseteq \forall es_originado.Nacimiento;$
  $Nacimiento \sqsubseteq Manantial;$
  $es_originado \equiv origina^-;$
  $Manantial \sqsubseteq \exists origina.Chortal$

We propose to add the reverse axiom of the C_2 definition and follow the recommendations of SOS and UE.

$$\left.\begin{array}{r} C_2 \sqsubseteq \exists R^-.C_{1a}; \cancel{C_{1b} \sqsubseteq \forall R.C_3;} \\ Disj(C_2, C_3); C_{1a} \sqsubseteq^* C_{1b}; \end{array}\right\} \Rightarrow \quad C_{1b} \sqsubseteq \forall R.(C_2 \sqcup C_3);$$
$$\left.\begin{array}{r} C_2 \sqsubseteq \forall R^-.C_{1a}; \cancel{C_{1b} \sqsubseteq \exists R.C_3;} \\ Disj(C_2, C_3); C_{1a} \sqsubseteq^* C_{1b}; \end{array}\right\} \Rightarrow \quad C_{1b} \sqsubseteq \exists R.(C_2 \sqcup C_3); \qquad (5.18)$$

Because of this recommendation, the correction of examples would be:

- $Mar \sqsubseteq \exists alimenta.Albufera;$
  $Albufera \sqsubseteq Laguna;$
  $Laguna \sqsubseteq Aguas\_Quietas\_Naturales;$
  $alimenta \equiv es\_alimentada^-;$ [**]

[§] R^- is an inverse property of R or we use an axiom $R^- inverseOf R$ in Protégé.

[¶] C_{1a} and C_{1b} could be the same class, or they have subset relations.

[‖] On the Protégé, you may find it as *alimenta inverseOf es_alimentada* or *es_alimentada inverseOf alimenta*.

[**] On the Protégé, you may find it as *alimenta inverseOf es_alimentada* or *es_alimentada inverseOf alimenta*.

$Aguas\_Quietas\_Naturales \sqsubseteq \forall es\_alimentada.(Mar$
$\sqcup\ Aguas\_Corrientes\_Naturales)$

- $R\acute{i}o \sqsubseteq \forall es_originado.Nacimiento$;
  $Nacimiento \sqsubseteq Manantial$;
  $es_originado \equiv origina^{-}$;
  $Manantial \sqsubseteq \exists origina.(R\acute{i}o \sqcup Chortal)$

This antipattern is more complex than UEWPI. Instead of *subPropertyOf* axiom, we have an *inverseOf* axiom and we take into account all subClass relations involved in this antipattern. We provide an example implementation for the first type of this antipattern (Code 20 and Code 21) . The second type is symmetric to the first one by exchanging ∃ and ∀.

Code 20 Pseudocode for UEWIP (5.17)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
  let $(C_2 \sqsubseteq superClass) = axiom$
  if $superClass$ is not $\exists$ restriction
    continue to the next loop
  let $\exists R^{-}.C_{1a} = superClass$
  if $C_{1a}$ is not a class
    continue to the next loop
  $inversePropertiesR$ := get all inverse properties of $R^{-}$
  if $|inversePropertiesR| == 0$
    continue to the next loop
  $ancestorClassesC1a$ := get all ancestor classes of $C_{1a}$
  for each $axiom2$ in $subClassAxioms$
    let $(C_{1b} \sqsubseteq superClass) = axiom2$
    if $superClass$ is not $\forall$ restriction
      continue to the next loop

    ...(continue to the next page)
  ...
  let $\forall R.C_3 = superClass$
  if $R \notin inversePropertiesR$
    continue to the next loop
  if $C_{1a} \neq C_{1b}$ and $C_{1b} \notin ancestorClassesC1a$
    continue to the next loop
  if $C_3 == C_2$ or neither $C_3$ is a class nor UnionOf
    continue to the next loop
  if $Disj(C_2, C_3)$
    $inversePropertyAxiom$ := get the axiom performing $R^{-} inverseOfR$
    $ancestorAxioms$ := get the axiom performing $C_{1a} \sqsubseteq^* C_{1b}$
    $disjointAxioms$ := get all axioms performing
                        the disjointness of $C_2$ and $C_3$
    $axiomList$ :=[$axiom1$] + $ancestorAxioms$+
                  [$inversePropertyAxiom, axiom2$] + $disjointAxioms$
    $paramList$ := [$C_{1b}, C_2, C_3, R, axiom2$]
    add pair ($axiomList, paramList$) into $results$
return $results$
```

Code 21 Pseudocode for UEWIP (5.18)

```
Load var [$C_{1b}, C_2, C_3, R, axiom2$] from detection process
$removedAction$ := create an action to remove $axiom2$
$newAxiom$ := $C_2 \sqsubseteq \forall R.(C_2 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction$, $addedAction$]
```

5.1.8 AntiPatterns hasValueisOneValue (VOV)

$$\begin{aligned} C_3 \sqsubseteq^* C_1; &\quad C_3 \sqsubseteq^* C_2; \\ C_1 \sqsubseteq\ hasValue\ R.\{v_1\}; &\quad C_2 \sqsubseteq\ hasValue\ R.\{v_2\}; \\ v_1 \neq v_2; &\quad R\ is\ functional; \end{aligned} \tag{5.19}$$

The $hasValue$ constraint is a built-in OWL property that links a restriction class to a value v_1, which can be either an individual or a data value. A restriction containing a $hasValue$ constraint describes a class of all individuals for which the property concerned has at least one value semantically equal to v_1 (it may have other values as well like v_2)[2].

OWL functional properties indicate how many times a property can be used for a given individual. The R property is functional means that each individual of the class C_3 has at most one value by the property R. For example, the *hasBirthday* relation between a person and his or her birthday is functional. Everyone has just one birthday [2].

Thus the ontology developer adds a *hasValue* restriction to a class using a functional property without remembering that there was already an *hasValue* restriction in the same class or in a parent class, respectively. The following are two examples of this antipattern in the SweetNumeric ontology:

- $Continental\_Margin \sqsubseteq^* GeometricalObject\_2D$;
  $Continental\_Margin \sqsubseteq^* GeometricalObject\_3D$;
  $GeometricalObject\_2D \sqsubseteq hasValue\ hasDimension.\{2\}$;
  $GeometricalObject\_3D \sqsubseteq hasValue\ hasDimension.\{3\}$;
  $hasDimension\ is\ functional$;

- $Aulacogen \sqsubseteq^* GeometricalObject\_2D$;
  $Aulacogen \sqsubseteq^* GeometricalObject\_3D$;
  $GeometricalObject\_2D \sqsubseteq hasValue\ hasDimension.\{2\}$;
  $GeometricalObject\_3D \sqsubseteq hasValue\ hasDimension.\{3\}$;
  $hasDimension\ is\ functional$;

The proposal for avoiding this antipattern is to remove the functional property of R.

$$\begin{aligned} C_3 \sqsubseteq^* C_1; &\qquad C_3 \sqsubseteq^* C_2; \\ C_1 \sqsubseteq\ hasValue\ R.\{v_1\}; &\qquad C_2 \sqsubseteq\ hasValue\ R.\{v_2\}; \\ v_1 \neq v_2; &\qquad \cancel{R\ is\ functional;} \end{aligned} \tag{5.20}$$

Applying this recommendation into the example, we just need to remove the axiom $hasDimension\ is\ functional$;.

An implementation of this antipattern is presented by Code 22 and Code 23. One special thing occurs in the Code 22 where a functional property can generate more than one pattern. Examples above show this event. Removing functionality of the property on a pattern will solve all inconsistencies on this antipattern with this property.

Code 22 Pseudocode for VOV (5.19)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom$
    if $superClass$ is not $hasValue$ restriction
      continue to the next loop
    let $hasValueR.v_1 = superClass$
    if $R$ is not functional data property
      continue to the next loop
    for each $axiom2$ in $subClassAxioms$
        let $(C_2 \sqsubseteq superClass) = axiom2$
        if $superClass$ is not $hasValue$ restriction
          continue to the next loop
        let $hasValueR_2.v_2 = superClass$
        if $R \neq R_2$ or $v_1 == v_2$
          continue to the next loop
        $isConnected := false$, defines whether $C_1$ and $C_2$ are connected
                                                    by chains of $\sqsubseteq$ or $\equiv$
        $axiomPath := []^*$, is a list of axiom that connects $C_1$ and $C_2$
        $intersectionDescendants = []$, is a set of intersection descendant
                                                between $C_1$ and $C_2$
        if $C_1 == C_2$
          $isConnected := true$
        else if $C_1 \sqsubseteq^+ C_2$ or $C_2 \sqsubseteq^+ C_1$ holds
              isConnected := true
              $axiomPath$ := get the axiom performing $C_1 \sqsubseteq^+ C_2$ or $C_2 \sqsubseteq^+ C_1$
        else
            $clsDescs1$ := get all descendent classes of $C_1$
            $clsDescs2$ := get all descendent classes of $C_2$
            $intersectionDescendants$ := intersection of $clsDescs1$ and $clsDescs2$
            $isConnected := (|intersectionDescendants| > 0)$
        if $isConnected$
          $functionalAxiom$ := get the functional data property axiom of $R$
          if $|intersectionDescendants| == 0$
            $axiomList := [functionalAxiom, axiom1, axiom2] + axiomPath$
            $paramList := [functionalAxiom]$
            add pair $(axiomList, paramList)$ into $results$
          else
```

...(continue to the next page)

```
        ...
        for each class $cls$ in $intersectionDescendants$
            if super class of $cls$ also in $intersectionDescendants$
                continue to the next loop
            $axiomPath$ := get the axiom performing $cls \sqsubseteq^+ C_1$ and $cls \sqsubseteq^+ C_2$
            $axiomList$ := $[functionalAxiom, axiom1, axiom2] + axiomPath$
            $paramList$ := $[functionalAxiom]$
            add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 23 Pseudocode for VOV (5.20)

```
Load var $[functionalAxiom]$ from detection process
$removedAction$ := create an action to remove $functionalAxiom$
return $[removedAction]$
```

5.1.9 AntiPattern EquivalenceIsDifference (EID)

$$C_1 \equiv C_2; Disj(C_1, C_2); \tag{5.21}$$

This pattern, which is only common for ontology developers with no previous training in OWL modeling, comes from the fact that the ontology developer means that C_1 is a subclass of C_2, or vice versa, but at the same time it is different from C_2 since he has more information. After a short training session the developer would discover that he really wants to express $C1 \sqsubseteq C2$. The followings are two of six of this antipattern in HydrOntology:

- $Cascada \equiv Catarata; Disj(Cascada, Catarata);$
- $Raudal \equiv Rápido; Disj(Raudal, Rápido);$

We propose to ask the ontology developer whether he really wants to define a synonym or a subclass-of relation. Depending on the ontology developer's answer, the equivalent axiom should be transformed into a subclass-of one or the less used concept should be suppressed according to the SOE recommendations. In order

*[] is the empty list.

to count less used concept, we use term frequency of a class that is calculated according to the appearance of the class in the all axioms in the ontology.

$$\cancel{C_1 \equiv C_2; Disj(C_1, C_2);} \Rightarrow C_1 \sqsubseteq C_2; \tag{5.22}$$

$$\Rightarrow C_2 \; label \; of C_1; \tag{5.23}$$

For the first example above, *Cascada* and *Catarata* respectively have term frequency 32 and 26. Some possibilities of condition after applying recommendations would be:

- $Cascada \equiv Catarata$;
- $Cascada \sqsubseteq Catarata$;
- $Catarata \sqsubseteq Cascada$;
- *Cascada* has some additional labels from *Catarata* as follows:
  - [*Comment*] : *Cascada o salto grande de agua*
  - [*Source*] : *Diccionario de la Real Academio de Española*

An implementation of the detection process is the Code 24. those recommendations, as we have shown in the correction of examples above, yield some action processes on the Code

Code 24 Pseudocode for EID (5.21)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$equivalentAxioms$ is a set of equivalent axiom in the ontology
for each $axiom$ in $equivalentAxioms$
    let $(C_1 \equiv C_2) = axiom$
    if $C_2$ is not a class
        continue to the next loop
    if $Disj(C_1, C_2)$
        $disjointAxioms$ := get all axioms performing the disjointness of $C_1$ and $C_2$
        $freq1$ := count the frequency of concept $C_1$ used
        $freq2$ := count the frequency of concept $C_2$ used
        $annotationAxiom1$ := create annotation axiom on $C_1$
                              with comment $Termfrequency : [freq1]$
        $annotationAxiom2$ := create annotation axiom on $C_2$
                              with comment $Termfrequency : [freq2]$
        $axiomList$ :=[$axiom, annotationAxiom1, annotationAxiom2$]
                      $+disjointAxioms$
        $disjointAxiom$ := a disjoint axiom from $disjointAxioms$ ††
        $paramList$ := $[C_1, C_2, axiom, disjointAxiom, freq1, freq2]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 25 Pseudocode for EID (5.22)

```
Load var $[C_1, C_2, axiom, disjointAxiom, freq1, freq2]$ from detection process
$removedAction$ := create an action to remove $disjointAxiom$
return [$removedAction$]
```

Code 26 Pseudocode for EID (5.22)

```
Load var $[C_1, C_2, axiom, disjointAxiom, freq1, freq2]$ from detection process
$removedAction1$ := create an action to remove $axiom$
$removedAction2$ := create an action to remove $disjointAxiom$
$newAxiom$ := $C_1 \sqsubseteq C_2$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction1$,$removedAction2$, $addedAction$]
```

††There is exactly one disjoint axiom in list of axiom *disjointAxioms*.

Code 27 Pseudocode for EID (5.23)

```
Load var $[C_1, C_2, axiom, disjointAxiom, freq1, freq2]$ from detection process
$actionList := listofaction$
if $freq1 \neq freq2$
  $removedAction$ := create an action to remove $axiom$
  add $removedAction$ into $actionList$
$removedAction$ := create an action to remove $disjointAxiom$
add $removedAction$ into $actionList$
if $freq1 > freq2$
  $removedAction$ := create an action to remove class $C_2$
  add $removedAction$ into $actionList$
  $annotationAxioms$ := get all annotation axiom of class $C_2$
  for each $annAxiom$ in $annotationAxioms$
      $addedAction$ := create an action to add the annotation axiom $annAxiom$
                       into $C_1$
      add $addedAction$ into $actionList$
else
  $removedAction$ := create an action to remove class $C_1$
  add $removedAction$ into $actionList$
  $annotationAxioms$ := get all annotation axiom of class $C_1$
  for each $annAxiom$ in $annotationAxioms$
      $addedAction$ := create an action to add the annotation axiom $annAxiom$
                       into $C_2$
      add $addedAction$ into $actionList$
return $actionList$
```

5.1.10 AntiPattern EquivalencesAreDifferences (EAD)

$$C_1 \equiv C_3; C_2 \equiv C_3; Disj(C_1, C_2); \tag{5.24}$$

The ontology developer has added a disjointness without remembering that he has already defined both classes having synonym to another same class that could be an anonymous class. This antipattern appeared 3 times in Tambis debugging process.

- $metal \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;
  $nonmetal \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;

- $metal \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;
  $metalloid \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;

- $metalloid \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;
  $nonmetal \equiv chemical \sqcap (\exists atomic\text{-}number.integer) \sqcap (= 1 atomic\text{-}number.\top))$;

We propose to ask the ontology developer whether he really wants to define a synonym or change definition of synonym for C_1 or C_2. It depends on the ontology developer's answer, if he prefers to define synonyms like current conditions, we simply recommend to remove the disjoint axiom.

$$C_1 \equiv C_3; C_2 \equiv C_3; \cancel{Disj(C_1, C_2)}; \tag{5.25}$$

You will find an implementation of this antipattern on the Code 28 and Code 29.

Code 28 Pseudocode for EAD (5.24)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$equivalentAxioms$ is a set of equivalent axiom in the ontology
for each $axiom1$ in $equivalentAxioms$
    let $(C_1 \equiv C_3) = axiom1$
    $equivalentAxioms2$ := get all equivalent axiom whose the second operand of
                           $\equiv$ is $C_3$
    for each $axiom2$ in $subClassAxioms2$
    let $(C_2 \equiv C_3) = axiom2$
    if $Disj(C_1, C_2)$
        $disjointAxioms$ := get all axioms performing
                            the disjointness of $C_1$ and $C_2$
        $axiomList := [axiom1, axiom2] + disjointAxioms$
        $disjointAxiom := adisjointaxiomfrom$disjointAxioms
        $paramList := [disjointAxiom]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 29 Pseudocode for EAD (5.25)

```
Load var $[disjointAxiom]$ from detection process
$removedAction$ := create an action to remove $disjointAxiom$
return $[removedAction]$
```

5.1.11 AntiPattern SubclassIsDifference (SID)

$$C_1 \sqsubseteq C_2; Disj(C_1, C_2); \tag{5.26}$$

This pattern comes from a misunderstanding of subclass-of relation. It is very closed to the EID one. This pattern was found once in the ontology Ontologia_Forestal.

- $Especies\_Forestals \sqsubseteq Recursos\_Forestal$;
- $Forest\_Species \sqsubseteq Forest\_Ressources$;

We propose to confirm whether the ontology developer really wants to define a subclass-of relation. Depending on the ontology developer's answer, the disjoint axiom should be suppressed or to follow the EID recommendations.

$$C_1 \sqsubseteq C_2; \cancel{Disj(C_1, C_2);} \tag{5.27}$$

An implementation of the detection process is the Code 30. The action process follows the action process of antipattern EAD (see Code 29).

Code 30 Pseudocode for SID (5.26)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom$ in $subClassAxioms$
    let $(C_1 \sqsubseteq C_2) = axiom$
    if $Disj(C_1, C_2)$
        $disjointAxioms$ := get all axioms performing
                             the disjointness of $C_1$ and $C_2$
        $axiomList := [axiom] + disjointAxioms$
        $disjointAxiom$ := a disjoint axiom from $disjointAxioms$
        $paramList := [disjointAxiom]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

5.1.12 AntiPattern SubclassesAreDifferences (SAD)

$$C_1 \sqsubseteq C_2; C_1 \sqsubseteq C_3; Disj(C_2, C_3); \tag{5.28}$$

The ontology developer has added a disjointness without remembering that he has already defined a class be subclass of both classes which are disjoint. The following is an example of this antipattern in HydOntology:

- $Zona\_Húmeda \sqsubseteq Humedal$;
  $Zona\_Húmeda \sqsubseteq Surgencia\_Natural$;

This antipattern is almost similar to the antipattern SID, but it uses two subclass-of relations. We propose to ask the ontology developer whether he really wants to define a subclass-of relation. Depending on the ontology developer's answer, the disjoint axiom should be suppressed or the the EID recommendations should be followed.

$$C_1 \sqsubseteq C_2; C_1 \sqsubseteq C_3; \cancel{Disj(C_2, C_3)}; \tag{5.29}$$

An implementation of the detection process is the Code 31. Like antipattern SID, the action process also follows the action process of antipattern EAD (see Code 29).

Code 31 Pseudocode for SAD (5.28)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq C_2) = axiom1$
    if neither $C_2$ is a class nor UnionOf
        continue to the next loop
    $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_1$
    for each $axiom2$ in $subClassAxioms2$
        let $(C_1 \sqsubseteq C_3) = axiom2$
        if $C_3 == C_2$ or neither $C_3$ is a class nor UnionOf
            continue to the next loop
        if $Disj(C_2, C_3)$
            $disjointAxioms$ := get all axioms performing
                                the disjointness of $C_2$ and $C_3$
            $axiomList := [axiom1, axiom2] + disjointAxioms$
            $disjointAxiom$ := a disjoint axiom from $disjointAxioms$
            $paramList := [disjointAxiom]$
            add pair $(axiomList, paramList)$ into $results$
return $results$
```

5.1.13 AntiPattern MinimalMaximalCardinalityRestriction (MMCaR)

$$C_1 \sqsubseteq \geq xR.C_2; C_3 \sqsubseteq = yR.C_2; with\ y < x; \tag{5.30}$$
$$C_1 \sqsubseteq \leq xR.C_2; C_3 \sqsubseteq = yR.C_2; with\ x < y; \tag{5.31}$$

C_1 and C_3 on both equations above must fulfill one of following conditions:

- C_1 and C_3 are the same class
- $C_1 \sqsubseteq^+ C_3$
- $C_3 \sqsubseteq^+ C_1$
- $C_4 \sqsubseteq^+ C_1$ and $C_4 \sqsubseteq^+ C_3$

The ontology developer may miss that a cardinality restriction about C_1, C_2, C_3 and the property R does exist. This antipattern does not appear in the debugged ontologies, it was deduced from the use of existential restriction that implies a minimal cardinality one. Thus it was derived from a more complicated one of the ECR 5.44, which appear several times in Hydrontology. The following is an example of this antipattern in ComputerScience:

- $TeachingFaculty \sqsubseteq \leq 3 takesCourse.Thing;$
  $LecturerTaking4Courses \sqsubseteq = 4 takesCourse.Thing;$
  $LecturerTaking4Courses \sqsubseteq Lecturer;$
  $Lecturer \sqsubseteq TeachingFaculty;$

We propose to ask the ontology developer which cardinality restriction is the good one and remove the other.

$$C_1 \sqsubseteq \geq xR.C_2; \cancel{C_3 \sqsubseteq = yR.C_2}; \tag{5.32}$$

$$C_1 \sqsubseteq \leq xR.C_2; \cancel{C_3 \sqsubseteq = yR.C_2}; \tag{5.33}$$

An implementation of the detection process is the Code 32. Any recommendations will remove an axiom as returned on $paramList := [axiom1, axiom2]$ in that code. Thus, we reuse the same action processes on the Code

Code 32 Pseudocode for MMCaR (5.31)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom1$
    if $superClass$ is not a $\leq$ restriction *
      continue to the next loop
    let $\leq xR.C_2 = superClass$
    for each $axiom2$ in $subClassAxioms$, but different with $axiom1$
        let $(C_3 \sqsubseteq superClass) = axiom2$
        if $superClass$ is not an $=$ restriction
          continue to the next loop
        let $= yR_2.C_{2b} = superClass$
        if $C_{2b} \neq C_2$ or $R \neq R_2$
          continue to the next loop
        if $x \leq y$
          continue to the next loop
        $isConnected := false$, defines whether $C_1$ and $C_3$ are connected
                                                   by chains of $\sqsubseteq$ or $\equiv$
        $axiomPath := []$, is a list of axiom that connects $C_1$ and $C_3$
        $intersectionDescendants := []$, is a set of intersection descendent
                                              between $C_1$ and $C_3$
        if $C_1 == C_3$
          isConnected := true
        else if $C_1 \sqsubseteq^+ C_3$ or $C_3 \sqsubseteq^+ C_1$ holds
              isConnected := true
              $axiomPath :=$ get the axiom performing $C_1 \sqsubseteq^+ C_3$ or $C_3 \sqsubseteq^+ C_1$
        else
            $clsDescs1 :=$ get all descendent classes of $C_1$
            $clsDescs3 :=$ get all descendent classes of $C_3$
            $intersectionDescendants :=$ intersection of $clsDescs1$ and $clsDescs3$
            $isConnected := (|intersectionDescendants| > 0)$
        if $isConnected$
          if $|intersectionDescendants| == 0$
            $axiomList := [axiom1, axiom2] + axiomPath$
            $paramList := [axiom1, axiom2]$
            add pair $(axiomList, paramList)$ into $results$
          else
              for each class $cls$ in $intersectionDescendants$
                  if super class of $cls$ also in $intersectionDescendants$
                    continue to the next loop
                  $axiomPath :=$ get the axiom performing
                                 $cls \sqsubseteq^+ C_1$ and $cls \sqsubseteq^+ C_3$
                  $axiomList := [axiom1, axiom2] + axiomPath$
                  $paramList := [axiom1, axiom2]$
                  add pair $(axiomList, paramList)$ into $results$
return $results$
```

5.1.14 AntiPattern Existential&CardinalityRestrictionWithInverseProperty (ECRWIP)

Basic formula for this antipattern as follow:

$$C_1 \sqsubseteq \exists R^{-}.C_2; C_2 \sqsubseteq = 1R.\top; C_2 \sqsubseteq \exists R.C_3; Disj(C_1, C_3); \tag{5.34}$$

From the basic formula above, we have found several extended formulas for this antipattern as follow:

- $C_1 \sqsubseteq \exists R^{-}.C_2; C_{2a} \sqsubseteq = 1R.\top; C_{2a} \sqsubseteq \exists R.C_3; Disj(C_1, C_3); C_2 \sqsubseteq^{+} C_{2a};$
- $C_1 \sqsubseteq \exists R^{-}.C_2; C_{2a} \sqsubseteq = 1R.\top; C_{2a} \sqsubseteq \exists S.C_3; Disj(C_1, C_3); C_2 \sqsubseteq^{+} C_{2a}; S \sqsubseteq R$

This antipattern appeared three times in HydrOntology debugging process. Following is an example of this antipattern.

- $Tuber\acute{i}a \sqsubseteq \exists alimenta.Fuente\_Artificial;$
  $es\_alimentada = alimenta^{-};$
  $Fuente\_Artificial \sqsubseteq = 1 es\_alimentada.\top;$
  $se\_extrae \sqsubseteq es\_alimentada;$
  $Fuente\_Artificial \sqsubseteq \exists se\_extrae.Acu\acute{i}fero;$
  $Disj(Tuber\acute{i}a, Acu\acute{i}fero);$

This antipattern leads to the unsatisfiability of C_2 on the equation 5.34 because $C_1 \sqsubseteq \exists R^{-}.C_2; \models C_2 \sqsubseteq \exists R.C_1;$. Thus we obtained a SOS antipattern and a MMCar one that composed an ECR one. Therefore, you should follow the SOS recommendation and after checking the cardinality restriction.

$$\left.\begin{array}{r} C_1 \sqsubseteq \exists R^{-}.C_2; C_2 \sqsubseteq = 1R.\top; \\ \cancel{C_2 \sqsubseteq \exists R.C_3;} Disj(C_1, C_3); \end{array}\right\} \Rightarrow \quad C_2 \sqsubseteq \exists R.(C_1 \sqcup C_3); \tag{5.35}$$

The correction of the above example after recommendation, would be:

- $Tuber\acute{i}a \sqsubseteq \exists alimenta.Fuente\_Artificial;$
  $es\_alimentada = alimenta^{-};$
  $Fuente\_Artificial \sqsubseteq = 1 es\_alimentada.\top;$
  $se\_extrae \sqsubseteq es\_alimentada;$
  $Fuente\_Artificial \sqsubseteq \exists se\_extrae.(Tuber\acute{i}a \sqcup Acu\acute{i}fero);$
  $Disj(Tuber\acute{i}a, Acu\acute{i}fero);$

[*] $\leq, \geq, =$ restriction mean respectively a minimal, maximal and exact cardinality restriction.

This antipattern is more complex than UEWIP. We have an exact cardinality restriction and possibility that sub property relation may occur. We also take into account all subClass relations involved in this antipattern. We provide an example implementation of this antipattern (Code 33 and Code 34).

Code 33 Pseudocode for ECRWIP (5.34)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom1$
    if $superClass$ is not $\exists$ restriction
        continue to the next loop
    let $\exists R^{-}.C_2 = superClass$
    if $C_2$ is not a class
        continue to the next loop
    $inversePropertiesR$ := get all inverse properties of $R^{-}$
    if $|inversePropertiesR| == 0)$
        continue to the next loop
    $ancestorClassesC2$ := get all ancestor classes of $C_2$
    for each $axiom2$ in $subClassAxioms$
        let $(C_{2b} \sqsubseteq superClass) = axiom2$
        if $superClass$ is not an exact cardinality restriction
            continue to the next loop
        let $= nR.C = superClass$
        if $n \neq 1$
            continue to the next loop
        if $C$ is not OWL Thing
            continue to the next loop
        if $R \notin inversePropertiesR$
            continue to the next loop
        if $C_{2b} \neq C_2$ and $C_{2b} \notin ancestorClassesC2$
            continue to the next loop
        $subPropertiesR$ :=get all sub property of $R$
        add $R$ into $subPropertiesR$
        $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_{2b}$
        for each $axiom3$ in $subClassAxioms$
            let $(C_{2b} \sqsubseteq superClass) = axiom3$

            ...(continue to the next page)
```

```
   ...
   if $superClass$ is not $\exists$ restriction
      continue to the next loop
   let $\exists S.C_3 = superClass$
   if $S \notin subPropertiesR$
      continue to the next loop
   if neither $C_3$ is a class nor UnionOf
      continue to the next loop
   if $Disj(C_1, C_3)$
      $disjointAxioms$ := get all axioms performing
                          the disjointness of $C_1$ and $C_3$
      $inversePropertyAxiom$ := get the axiom performing
                                $R^{-} inverseOf R$
      $ancestorAxioms$ := get the axiom performing $C_2 \sqsubseteq^{*} C_{2b}$
      $subPropertyAxiom$ := get the axiom performing
                            $S\ subPropertyOf\ R$ if possible
      $axiomList$ :=$[axiom1] + ancestorAxioms + [inversePropertyAxiom] +$
                    $[axiom2, subPropertyAxiom, axiom3] + disjointAxioms$
      $paramList$ := $[C_1, C_{2b}, C_3, S, axiom3]$
      add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 34 Pseudocode for ECRWIP (5.35)

```
Load var $[C_1, C_{2b}, C_3, S, axiom3]$ from detection process
$removedAction$ := create an action to remove $axiom3$
$newAxiom$ := $C_{2b} \sqsubseteq \exists R.(C_1 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction$, $addedAction$]
```

5.1.15 AntiPattern SumOfSomwithExactRestriction (SOSER)

Basic formula for this antipattern as follow:

$$C_1 \sqsubseteq = 1R.\top; C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubset \exists R.C_3; Disj(C_2, C_3); \tag{5.36}$$

The ontology developer has added restrictions about C_1, C_2 and C_3 using a property, exact restriction with cardinality one and a disjoint axiom. Obviously, this antipattern is a specialization of SOS. The additional axiom $C_2 \sqsubseteq = 1R.\top;$ causes C_1 be unsatisfiable.

From the basic formula above, we have found several extended formulas for this antipattern.

- $C_1 \sqsubseteq = 1R.\top; C_{1a} \sqsubseteq \exists R.C_2; C_{1b} \sqsubseteq \exists R.C_3; Disj(C_2, C_3);$, where C_1, C_{1a} and C_{1b} must fulfill one of the following conditions:
  - $C_1 \sqsubseteq^* C_{1a}$ and $C_1 \sqsubseteq^* C_{1b}$
  - $C_{1a} \sqsubseteq^* C_1$ and $C_1 \sqsubseteq^* C_{1b}$
  - $C_{1b} \sqsubseteq^* C_1$ and $C_1 \sqsubseteq^* C_{1a}$
- $C_1 \sqsubseteq = 1R.\top; C_1 \sqsubseteq \exists R_1.C_2; C_1 \sqsubseteq \exists R_2.C_3; Disj(C_2, C_3);$, where $R_1 \sqsubseteq R$ and $R_2 \sqsubseteq R$.
- Combination between subclass and sub property from previous points above.

This antipattern appeared twice in HydrOntology debugging process. Following is an example of this antipattern.

- $Arroyo \sqsubseteq = 1es\_originado.\top;$
  $Aguas\_Corrientes\_Naturales \sqsubseteq \exists es\_originado.Manantial;$
  $Torrente \sqsubseteq \exists es\_originado.(Glaciar \sqcup Masa\_de\_Hielo);$
  $Torrente \sqsubseteq Arroyo;$
  $Arroyo \sqsubseteq Aguas\_Corrientes\_Naturales;$
  $Disj(Manantial, (Glaciar \sqcup Masa\_de\_Hielo));$
  , because $Disj(Manantial, Glaciar)$ and $Disj(Manantial, Masa\_de\_Hielo)$

Since this antipattern is a special case of SOS antipattern, then we follow the recommendations of SOS.

$$\left.\begin{array}{l} C_1 \sqsubseteq = 1R.\top; C_1 \sqsubseteq \exists R.C_2; \\ \cancel{C_1 \sqsubseteq \exists R.C_3;} Disj(C_2, C_3); \end{array}\right\} \Rightarrow \quad C_1 \sqsubseteq \exists R.(C_2 \sqcup C_3); \qquad (5.37)$$

The correction of the example would be:

- $Arroyo \sqsubseteq = 1es\_originado.\top;$
  $Aguas\_Corrientes\_Naturales \sqsubseteq \exists es\_originado.Manantial;$
  $Torrente \sqsubseteq \exists es\_originado.(Manantial \sqcup Glaciar \sqcup Masa\_de\_Hielo);$
  $Torrente \sqsubseteq Arroyo;$
  $Arroyo \sqsubseteq Aguas\_Corrientes\_Naturales;$
  $Disj(Manantial, (Glaciar \sqcup Masa\_de\_Hielo));$

This antipattern is almost as complex as ECRWIP, but we do not use an inverse property. We provide an example implementation of this antipattern (Code 35 and Code 36). This implementation has already covered basic and extended formula for the detection process. On the detection process, we suggest to choose an axiom containing a parent class (if there is a subclass relation) which will be removed on the action process.

Code 35 Pseudocode for SOSER (5.36)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom1$
    if $superClass$ is not an exact cardinality restriction
        continue to the next loop
    let $= nR.C = superClass$
    if $n \neq 1$
        continue to the next loop
    if $C$ is not OWL Thing
        continue to the next loop
    $subPropertiesR$ :=get all sub property of $R$
    add $R$ into $subPropertiesR$
    for each $axiom2$ in $subClassAxioms$
        let $(C_{1a} \sqsubseteq superClass) = axiom2$
        if $superClass$ is not $\exists$ restriction
            continue to the next loop
        let $\exists R_1.C_2 = superClass$
        if $R_1 \notin subPropertiesR$
            continue to the next loop
        if neither $C_2$ is a class nor UnionOf
            continue to the next loop
        for each $axiom3$ in $subClassAxioms$
            let $(C_{1b} \sqsubseteq superClass) = axiom3$
            if $superClass$ is not $\exists$ restriction
                continue to the next loop
            let $\exists R_2.C_3 = superClass$
            if $R_2 \notin subPropertiesR$
                continue to the next loop

            ...(continue to the next page)
```

```
   ...
   if neither $C_2$ is a class nor UnionOf
     continue to the next loop
   if $C_1 \sqsubseteq^* C_{1a}$ and $C_1 \sqsubseteq^* C_{1b}$ holds
     $bottomClass := C_1$
     $topClass1 := C_{1a}$
     $topClass2 := C_{1b}$
   else if $C_{1a} \sqsubseteq^* C_1$ and $C_{1a} \sqsubseteq^* C_{1b}$ holds
       $bottomClass := C_{1a}$
       $topClass1 := C_1$
       $topClass2 := C_{1b}$;
   else if $C_{1b} \sqsubseteq^* C_1$ and $C_{1b} \sqsubseteq^* C_{1a}$ holds
       $bottomClass := C_{1b}$
       $topClass1 := C_1$
       $topClass2 := C_{1a}$
   else
       continue to the next loop
   if $Disj(C_2, C_3)$
     $subPropertyAxioms :=$ get all axioms performing $R_1$ $subPropertyOf$ $R$
                           and $R_2$ $subPropertyOf$ $R$ if possible
     $disjointAxioms :=$ get all axioms performing
                        the disjointness of $C_2$ and $C_3$
     $ancestorAxioms1 :=$ get all axioms performing
                          $bottomClass \sqsubseteq^* topClass1$
     $ancestorAxioms2 :=$ get all axioms performing
                          $bottomClass \sqsubseteq^* topClass2$
     $axiomList :=[axiom1, axiom2, axiom3] + subPropertyAxioms+$
                  $disjointAxioms + ancestorAxioms1 + ancestorAxioms2$
     if $topClass1 == C_1$
       if $topClass2 == C_{1a}$
         $paramList := [C_2, C_3, C_{1a}, R_1, axiom2]$
       else
           $paramList := [C_2, C_3, C_{1b}, R_2, axiom2]$
     else if $topClass2 == C_1$
           if $topClass1 == C_{1a}$
             $paramList := [C_2, C_3, C_{1a}, R_1, axiom2]$
           else
               $paramList := [C_2, C_3, C_{1b}, R_2, axiom2]$
     else
               $paramList := [C_2, C_3, C_{1b}, R_2, axiom2]$
     add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 36 Pseudocode for SOSER (5.37)

```
Load var $[C_2, C_3, Class, Relation, axiom]$ from detection process
$removedAction$ := create an action to remove $axiom3$
$newAxiom := Class \sqsubseteq \exists Relation.(C_2 \sqcup C_3)$
$addedAction$ := create an action to add $newAxiom$
return $[removedAction, addedAction]$
```

5.2 Cognitive Logical AntiPatterns (CLAP)

5.2.1 AntiPattern SynonymOrEquivalence (SOE)

$$C_1 \equiv C_2; \tag{5.38}$$

The ontology developer wants to express that two classes C_1 and C_2 are identical. This is not very useful in a single ontology that does not import others. Indeed, what the ontology developer generally wants to represent is a terminological synonymy relation: the class C_1 has two labels: C_1 and C_2. Usually one of the classes is not used anywhere else in the axioms defined in the ontology. In HydrOntology, this antipattern appears six times The following is an example of this antipattern in HydrOntology:

- $Afluente \equiv R\acute{i}o$;

The proposal for avoiding this antipattern is the following (if C_2 is the less used term in the ontology) add all the comments and labels of C_2 into C_1 and remove C_2.

$$C_1 \not\equiv \not C_{\not 2} \not; \quad \Rightarrow \quad C_1.[rdfs : label|comment] = C_2.[rdfs : label|comment] \tag{5.39}$$

For the example above, $Afluente$ and $R\acute{i}o$ respectively have term frequency 24 and 146. The proposal of this antipattern will remove the class $Afluente$. $R\acute{i}o$ will have some additional labels from $Afluente$ as follows:

- $[Provenance]$: $Curso\ de\ agua\ principal\ -\ Catálogo\ de\ fenómenos.\ Proyecto\ GEOALEX$
- $[Provenance]$: $Directiva\ Marco\ del\ Agua.Unión\ Europea$
- $[Provenance]$: $Water\ Framework\ Directive.\ European\ Union$
- $[Comment]$: $Masa\ de\ agua\ continental\ que\ fluye\ en\ su\ mayor\ parte\ sobre\ la\ superficie\ del\ suelo,\ pero\ que\ puede\ fluir\ bajo\ tierra\ en\ parte\ de\ su\ curso$

- $[label]$: $Curso\ de\ agua\ principal$
- $[label]$: $Curso\ fluvial$
- $[label]$: $River$

An implementation of this antipattern is simpler than EID antipattern. The Code 37 represents the detection process of it, while the Code 38 represents the action process.

Code 37 Pseudocode for SOE (5.38)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$equivalentAxiomsOntology$ is a map of equivalent axiom and ontology in the ontologies
for each $(axiom, ontology)$ in $equivalentAxiomsOntology$
    let $(C_1 \equiv C_2) = axiom$
    if $C_2$ is not a class
        continue to the next loop
    if $C_1$ exists in $ontologies \backslash \{ontology\}$‡‡
        continue to the next loop
    if $C_2$ exists in $ontologies \backslash ontology$
        continue to the next loop
    $freq1$ := count the frequency of concept $C_1$ used
    $freq2$ := count the frequency of concept $C_2$ used
    $annotationAxiom1$ := create annotation axiom on $C_1$
                          with comment $Termfrequency : [freq1]$
    $annotationAxiom2$ := create annotation axiom on $C_2$
                          with comment $Termfrequency : [freq2]$
    $axiomList := [axiom, annotationAxiom1, annotationAxiom2]$
    $paramList := [C_1, C_2, axiom, freq1, freq2]$
    add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 38 Pseudocode for SOE (5.39)

```
Load var [$C_1, C_2, axiom, freq1, freq2$] from detection process
$actionList := listofaction$
if $freq1 \neq freq2$
  $removedAction$ := create an action to remove $axiom$
  add $removedAction$ into $actionList$
if $freq1 > freq2$
  $removedAction$ := create an action to remove class $C_2$
  add $removedAction$ into $actionList$
  $annotationAxioms$ := get all annotation axiom of class $C_2$
  for each $annAxiom$ in $annotationAxioms$
      $addedAction$ := create an action to add the annotation axiom $annAxiom$
                       into $C_1$
      add $addedAction$ into $actionList$
else
  $removedAction$ := create an action to remove class $C_1$
  add $removedAction$ into $actionList$
  $annotationAxioms$ := get all annotation axiom of class $C_1$
  for each $annAxiom$ in $annotationAxioms$
      $addedAction$ := create an action to add the annotation axiom $annAxiom$
                       into $C_2$
      add $addedAction$ into $actionList$
return $actionList$
```

5.2.2 AntiPattern SumOfSom (SOS)

$$C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq \exists R.C_3; Disj(C_2, C_3); \tag{5.40}$$

The ontology developer has added a new existential restriction without remembering that he has already defined another existential restriction for the same class and property. Although this could be no problem in some cases (e.g., a child has at least one mother and at least one father), in many cases it represents a modeling error. Moreover notice that this antipattern implied a minimal maximal cardinality restriction (MMCaR) $C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq \exists R.C_3; Disj(C_2, C_3); \models C_1 \sqsubseteq (\geq 2R.\top)$. When the antipattern is detected, we should check if any MMCaR antipattern occurred in order to produce an ECR one. The following is an example of this antipattern:

- $Rio \sqsubseteq \exists puede\_fluir.Corriente\_Subterránea;$
  $Rio \sqsubseteq \exists puede\_fluir.Ponor;$

Ontology developers should understand clearly the combination of two existential restrictions. Our proposal is to clarify the modeling thus we propose to merge the two axioms in one existential restriction using disjunction of C_2 and C_3.

$$\cancel{C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq \exists R.C_3;} Disj(C_2, C_3); \quad \Rightarrow \quad C_1 \sqsubseteq \exists R.(C_2 \sqcup C_3); \tag{5.41}$$

An implementation of this antipattern will follow the implementation of antipattern OIL (see Code 10), but instead of universal restrictions here, we use existential restrictions.

5.3 Guidelines

5.3.1 Guideline UnionInEquivalency(UIE)

$$C_1 \equiv C_1 \sqcup C_2; \tag{5.42}$$

The ontology developer may want to say that C_2 does not take any instances outside of C_1. Instead of defining C_1 as the equivalency of $C_1 \sqcup C_2$, it could be more appropriate to state that C_2 is a subclass of C_1. The following is an example of this antipattern in HydrOntology:

- $Deposito \equiv Deposito \sqcup Recinto$;

We propose a subClass relation as impact of equivalency formula 5.42.

$$\cancel{C_1 \equiv C_1 \sqcup C_2;} \Rightarrow C_2 \sqsubseteq C_1; \tag{5.43}$$

After applying the recommendation, the correction would be:

- $Recinto \sqsubseteq Deposito$;

We take into account the number of operands in the union. Thus, we need to classify whether an operand belongs to a group that each of element is equivalent with C_1 or not (see the Code 39). For every operand that is not equivalent with C_1, we construct a new subclass relation (see the Code 40).

Code 39 Pseudocode for UIE (5.42)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$equivalentAxioms$ is a set of equivalent axiom in the ontology
for each $axiom$ in $equivalentAxioms$
    let $(C_1 \equiv unionClass) = axiom$
    if $unionClass$ is not a union
        continue to the next loop
    if all operands in $unionClass$ are a class
                    continue to the next loop
    $equivalentClosure$ := get all class that equivalent with $C_1$
    $operandsEquiv$ := [], set of operands that are equivalent with $C_1$
    $operandsNot$ := [], set of operands that are not equivalent with $C_1$
    $equivalentAxioms$ := [], set of equivalent axiom
    for each $operand$ in operands of $unionClass$
        if $operand \in equivalentClosure$
            add $operand$ into $operandsEquiv$
            $equAxioms$ := get all axioms performing $operand \equiv *C_1$
            $equivalentAxioms := equivalentAxioms + equAxioms$
        else
                add $operand$ into $operandsNot$
    if $|operandsEquiv| > 0$ and $|operandsNot| > 0$
        $axiomList := [axiom] + equivalentAxioms$
        $paramList := [axiom, C_1, operandsNot]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 40 Pseudocode for UIE (5.43)

```
Load var $[axiom, C_1, operandsNot]$ from detection process
$removedAction$ := create an action to remove $axiom$
$addedActionList$ is a list
for each operand in operandsNot
    $newAxiom := operand \sqsubseteq C_1$
    $addedAction$ := create an action to add $newAxiom$
    add $addedAction$ into $addedActionList$
return $[removedAction] + addedActionList$
```

5.3.2 Guideline Existential & Cardinality Restriction(ECR)

$$C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq (\geq 2R.\top); (for\ example) \tag{5.44}$$

Ontology developers with little background in formal logic find difficult to understand that "only" does not imply "some" [18]. This antipattern is a counterpart of that fact. Developers may forget that existential restrictions contain a cardinality constraint: $C_1 \sqsubseteq \exists R.C_2 \models C_1 \sqsubseteq (\geq 1 R.C_2)$. Thus, when they combine existential and cardinality restrictions, they may be actually thinking about universal restrictions with those cardinality constraints. This antipattern can be a complex one because it may contain a SOS antipattern and a MMCaR one. The following is an example of this antipattern in HydrOntology:

- $Estero \sqsubseteq \exists está\_proxima.Desembocadura$
  $Estero \sqsubseteq \geq 1 está\_proxima.\top$

We propose to transform the existential restriction into a universal one when a cardinality restriction exists.

$$\cancel{C_1 \sqsubseteq \exists R.C_2;} C_1 \sqsubseteq (\geq 2R.\top); \quad \Rightarrow \quad C_1 \sqsubseteq \forall R.C_2; \tag{5.45}$$

Because of this proposal, the correction of the example would be:

- $Estero \sqsubseteq \forall está\_proxima.Desembocadura$
  $Estero \sqsubseteq \geq 1 está\_proxima.\top$

It is easy to create an implementation of this antipattern, since what we need has been implemented in the previous antipatterns. The Code 41 and 42 are an implementation of this antipattern.

Code 41 Pseudocode for ECR (5.44)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axiom in the ontology
for each $axiom1$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom1$
    if $superClass$ is not a $\geq$-restriction
      continue to the next loop
    let $\geq xR.C = superClass$
    if $C$ is not OWL Thing
      continue to the next loop
    $subClassAxioms2$ := get all subClassAxiom whose the subClass is $C_1$
    for each $axiom2$ in $subClassAxioms2$
        let $(C_1 \sqsubseteq superClass) = axiom2$
        if $superClass$ is not $\exists$ restriction
          continue to the next loop
        let $\exists R_2.C_2 = superClass$
        if $R \neq R_2$
          continue to the next loop
        $axiomList := [axiom1, axiom2]$
        $paramList := [C_1, C_2, R, axiom2]$
        add pair $(axiomList, paramList)$ into $results$
return $results$
```

Code 42 Pseudocode for ECR (5.45)

```
Load var $[C_1, C_2, R, axiom2]$ from detection process
$removedAction$ := create an action to remove $axiom2$
$newAxiom := C_1 \sqsubseteq \forall R.C_2$
$addedAction$ := create an action to add $newAxiom$
return [$removedAction$, $addedAction$]
```

5.3.3 Guideline Distributivity On Subclass (DOS)

$$C_1 \sqsubseteq C_{21} \sqcap ... \sqcap C_{2n}; \tag{5.46}$$

Sometimes, we cannot see a consistency because a axiom is inside of another axiom. The ontology developer has developed a complex axiom and he does not realize that it can be decomposed to several axioms. By applying distributivity on subclass relation over intersection, it will remove the original axiom and produce n new axioms that will make debugging process easier.

In HydrOntology, this antipattern appears three times.

- $Confluencia \sqsubseteq (\exists conecta.R\acute{\imath}o) \sqcap (= 2conecta.\top);$
- $Aguas\_Corrientes\_Naturales \sqsubseteq (\forall desemboca.(Aguas\_Corrientes$ $\sqcup Aguas\_Marinas \sqcup Aguas\_Quietas)) \sqcap (= 1desemboca.\top);$
- $Captaci\acute{o}n \sqsubseteq (\exists captura.R\acute{\imath}o) \sqcap (= 2captura.\top);$

We propose a recommendation below that semantically both sides are equivalent.

$$\cancel{C_1 \sqsubseteq C_{21} \sqcap ... \sqcap C_{2n};} \quad \Rightarrow \quad C_1 \sqsubseteq C_{21}; ...; C_1 \sqsubseteq C_{2n}; \tag{5.47}$$

We will get the correction of examples as follows:

- $Confluencia \sqsubseteq \exists conecta.R\acute{\imath}o;$
  $Confluencia \sqsubseteq = 2conecta.\top;$
- $Aguas\_Corrientes\_Naturales \sqsubseteq \forall desemboca.(Aguas\_Corrientes$ $\sqcup Aguas\_Marinas \sqcup Aguas\_Quietas);$
  $Aguas\_Corrientes\_Naturales \sqsubseteq = 1desemboca.\top;$
- $Captaci\acute{o}n \sqsubseteq \exists captura.R\acute{\imath}o;$
  $Captaci\acute{o}n \sqsubseteq = 2captura.\top;$

The detection process of this antipattern is implemented in very simple way on the Code 43. The action process (Code 44) follows the action process of antipattern UIE, but we have another direction of $\sqsubseteq$ for each operand.

Code 43 Pseudocode for DOS (5.46)

```
$results$ is a list of pair (list of axiom, list of parameter for action process)
$subClassAxioms$ is a set of subclass axioms in the ontology
for each $axiom$ in $subClassAxioms$
    let $(C_1 \sqsubseteq superClass) = axiom$
    if $superClass$ is an intersection
        add $([axiom], [axiom, C_1, operands])$ into $results$
return $results$
```

Code 44 Pseudocode for DOS (5.47)

```
Load var $[axiom, C_1, operands]$ from detection process
$removedAction$ := create an action to remove $axiom$
$addedActionList$ is a list
for each operand in operands
    $newAxiom := C_1 \sqsubseteq operand$
    $addedAction$ := create an action to add $newAxiom$
    add $addedAction$ into $addedActionList$
return $[removedAction] + addedActionList$
```

Chapter 6

Apero Plug-in

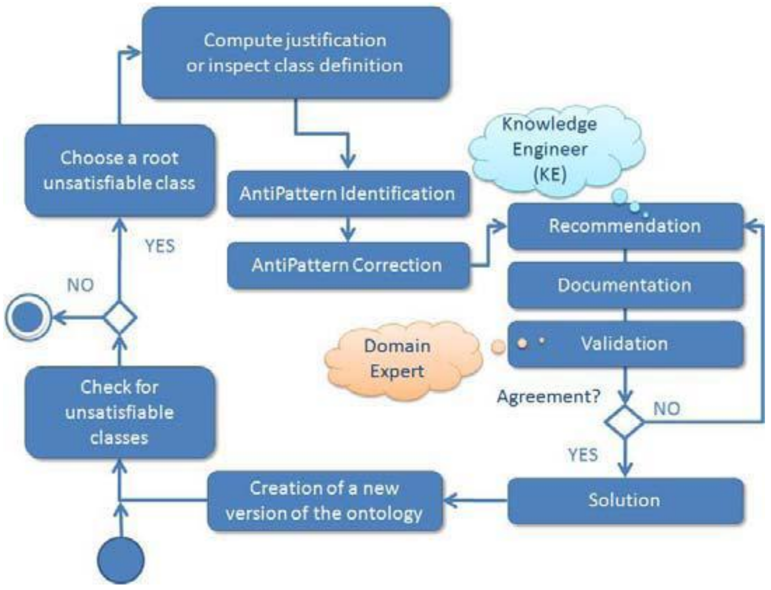
This chapter presents the support tool that we call Apero (AntiPatternExtRactiOn). Our discussion covers how to analyze, design and implement this tool.

6.1 Analysis

OWL ontology debugging features have been proposed in the literature with different degrees of formality ([9],[15],[23]). But, they are mainly focused on the explanations of logical entailments and are not so focused on the ontology engineering side. A debugging strategy that involves the ontology engineering side, has been proposed [5]. It leads an ontology developer how to debug incoherent ontologies. It has proposed a global ontology debugging life cycle involving the role of knowledge engineer and domain expert. We assume that an ontology developer has those roles. Figure 6.1 displays graphically this strategy.

From the workflow that is described on the figure, we propose Apero as a plug-in that we attach into Protégé which must have the following functionalities:

- Manual processes in the workflow are checking for unsatisfiable classes, choosing a root unsatisfiable classes, computing justification or inspecting class definition, documentation, validation and creation of a new version of the ontology. Apero must be compatible with all manual processes. Especially for documentation and validation, Apero must provide enough information to support those processes. The information should cover :
  - the formula of an antipattern
  - how an antipattern is constructed
  - available recommendations and their description
  - a set of actions that the ontology developer can execute


Compute justification or inspect class definition
Knowledge Engineer (KE)
Choose a root unsatisfiable class
AntiPattern Identification
AntiPattern Correction
Recommendation
Documentation
Validation
Domain Expert
YES
NO
Agreement?
NO
YES
Check for unsatisfiable classes
Creation of a new version of the ontology
Solution

- In order to support the manual process, especially inspecting class definitions, Apero will provide a transformation process that helps the ontology developer to convert a class definition into an expected one. He needs to save into a new ontology since the ontology before and after this process are unlike.
  The axiom transformation has several rules that we cannot classify them into an antipattern. For the current implementation, we will limit the implementation only to one rule transformation (formula 6.1).

$$C_1 \equiv C_{21} \sqcap ... \sqcap C_{2n}; \quad \Rightarrow \quad C_1 \sqsubseteq C_{21}; ...; C_1 \sqsubseteq C_{2n}; \qquad (6.1)$$

- Apero will identify or detect an antipattern.
  An antipattern is constructed by a set of axioms, as an instance of the antipattern. Two instances of an antipattern possibly can share axioms. We choose to repeat displaying the axiom, thus the ontology developer find it easier to see how an instance is built. The result of this process is a table of antipattern instances.

- Apero must able to do some corrections and show the result as a recommendation for the future solution.
  The ontology developer as the user probably wants to focus on a certain instance. Apero will provide the correction immediately for each observed instance, without affecting the current ontology.

- Apero will apply the selected recommendation as a solution.
  A recommendation consists of a set of actions adding or removing axioms

produced by correction of an antipattern.

- Apero facilitates users to propose and create new antipattern.
  Apero must be easy to be configured and allow users to propose new antipattern. They are also expected to be able to create new antipatterns as easy as possible.

- Apero will accommodate two type of antipattern implementations.
  We know that there are two types of antipatterns in our previous discussion. Both OPPL and Lint should be implemented transparently.

6.2 Design

6.2.1 Configuration

Configuration of Apero is a configuration that contains information needed by Apero. For every item of information in the configuration we call it as parameter. According to what we have already analyzed, we design the configuration as the following parameters:

- OPPL folder
  Since we will have two types of antipatterns, the configuration must accommodate these. An OPPL antipattern is provided as a script in a text file. Thus, we put all scripts into a specific folder that we parameterize $OPPLFolder$.

- Group of antipattern category This configuration covers information about all groups of antipattern category. It consists of number of group and group detail as parameters. A group has name, description and remark (see table 6.2.1).

Table 6.1: A group of antipattern category

No	Parameter	Optional	Summary
1	name		group name and must be unique
2	desc	Yes	Short description of this group
3	remark	Yes	Full description of this group

- Antipattern This configuration covers information about all antipatterns. It consists of number of antipattern and detail of antipattern. A antipattern configuration has information depending on the type of implementation, namely oppl and lint as we state in the table 6.2 and 6.3 respectively.

The validity of configuration is determined by its parameters with condition as follows:

Table 6.2: A OPPL-based antipattern configuration

No	Parameter	Optional	Summary
1	name		Antipattern name and must be unique
2	formula		Latex formula for this antipattern
3	formulaDetail	Yes	Additional latex formula if necessary
3	desc	Yes	Description of this antipattern
4	type		Type of this antipattern, either Lint or OPPL
5	opplQueryPrefix		file name that contains variable declaration and query part of OPPL script and represent a detection process
6	numberOfAction		Number of possible recommendation
7	group		Group name (see table 6.2.1)
For each action 1 to [numberOfAction]			
8	action[i].name		Action name and must be unique in this antipattern, [i] represents a sequence number of the action
9	action[i].query		file name that contains action part of OPPL Script and represents an action process
10	action[i].remark	Yes	Description of this action

Table 6.3: A Lint-based antipattern configuration

No	Parameter	Optional	Summary
1	name		Antipattern name and must be unique
2	formula		Latex formula for this antipattern
3	formulaDetail	Yes	Additional latex formula if necessary
3	desc	Yes	Description of this antipattern
4	type		Type of this antipattern, either Lint or OPPL
5	lintQueryClass		Java class that represents a detection process
6	numberOfAction		Number of possible recommendation
7	group		Group name (see table 6.2.1)
For each action 1 to [numberOfAction]			
8	action[i].name		Action name and must be unique in this antipattern, [i] represents a sequence number of the action
9	action[i].lintClass		Java class that represents an action process
10	action[i].remark	Yes	Description of this action

- Antipattern names must be unique.
- The total of existing antipattern in the configuration has be to equal or greater than the number of antipattern, with the correct sequence number.
- The total number of groups in the configuration has to be equal or greater than the number of groups, with the correct sequence number.
- Action names for every antipattern must be unique.

- The total number of actions for every antipattern has to be equal or greater than its number of actions, with the correct sequence number.
- The value of parameter *type* are only $OPPL$ (not case sensitive) and $lint$. Otherwise, we treat it as a Lint-based antipattern.
- A OPPL-based antipattern must have parameter $opplQueryPrefix$ and $action[i].query$.
- A Lint-based antipattern must have parameter $lintQueryClass$ and $action[i].lintClass$.
- The text file of parameter $opplQueryPrefix$ and $action[i].query$ must be exist in the folder parameter $OPPLFolder$.
- The Java class of parameter $lintQueryClass$ and $action[i].lintClass$ must be exist in the package plug-in.
- The value of parameter *group* for every antipattern has to be in one of existing groups.
- All parameters must have a value except an optional parameter.

6.2.2 Transformation Process Design

Graphically, we design the transformation process on Figure 6.1. We expect that Apero will find all axioms from the ontology that fulfill the existing transformation rules, and provide it together with its solutions (result axiom and annotation axiom). Besides getting the result axiom as result of the process, Apero will also create a new annotation axiom for every result axiom informing to the ontology developer that this result axiom is automatically generated by Apero. After he manually selects the transformation to proceed, Apero will transform or execute the selected transformation permanently into the ontology. He must save the current ontology as a new ontology since both ontologies before and after the process are not the same anymore.

6.2.3 Detection Process Design

The figure 6.2 displays a flowchart of the detection process. What we display on the flowchart is briefly how Apero detects an antipattern. Details of the detection process for every antipattern have been described with pseudocodes in Chapter 4 and 5.

The ontology developer probably wants to observe some antipatterns. After he chooses some antipatterns, for every selected antipattern, Apero will try to match with all axioms in the ontology. The matching axioms (including its

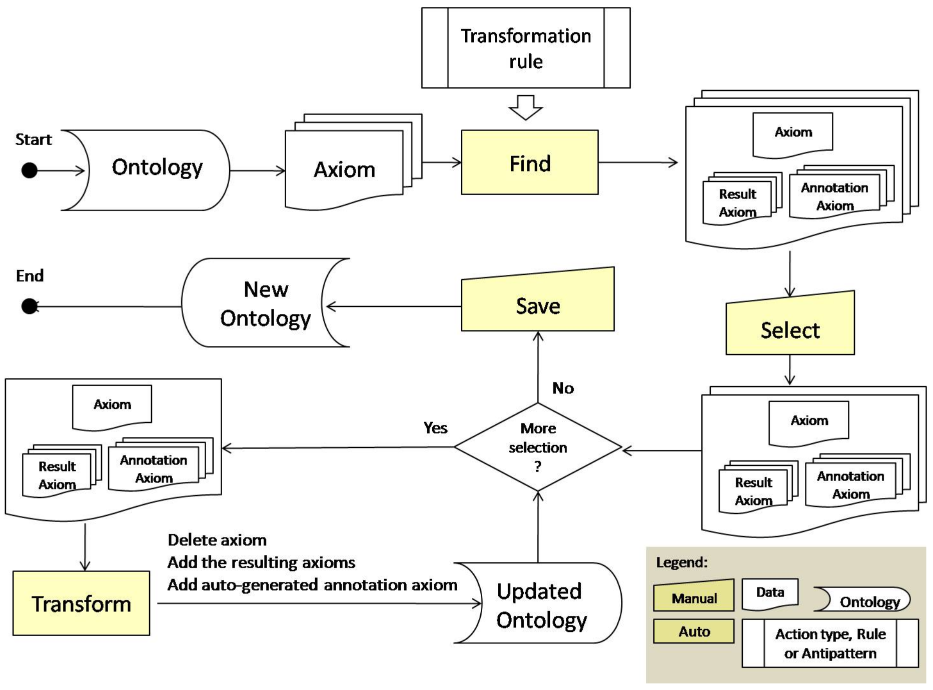


Figure 6.1: Transformation process flowchart

binding variables) with an antipattern is called as an instance of antipattern. It is important to keep pair of the matching axioms and its binding variables that we can use as parameters for execution process. Since we have two types of implementation (OPPL and Lint), Apero has two different detection process. In OPPL, a matched axiom is called an instantiated axiom.

OPPL naturally can evaluate a query (detection and action) once. Evaluation only simulates, does not affect to the ontology, and takes every instantiated axioms and its binding variables as an instance. Thus in order to complete a detection query, we always take one action query. Meanwhile, Lint sequentially takes Java class for detection process, runs it and returns pairs of matched axioms and its binding variables as instances. Apero will memorize every set of instance into a detection table, such that Apero will be easy and fast to display the result.

6.2.4 Execution Process Design

The figure 6.3 displays a flowchart of the execution process. This covers a lot of manual process as well as automatic process. We assume that the ontology developer is focusing on an antipattern. The flowchart briefly shows how Apero proceed with instances of antipatterns after the detection process. At one point in the flowchart, we will run the action process that we discussed in Chapter 4 and Chapter 5. This action process will be run in the box $EvaluatebyOPPL$ or $GetOWLActionListbyLint$. An action list as output of those processes is a list

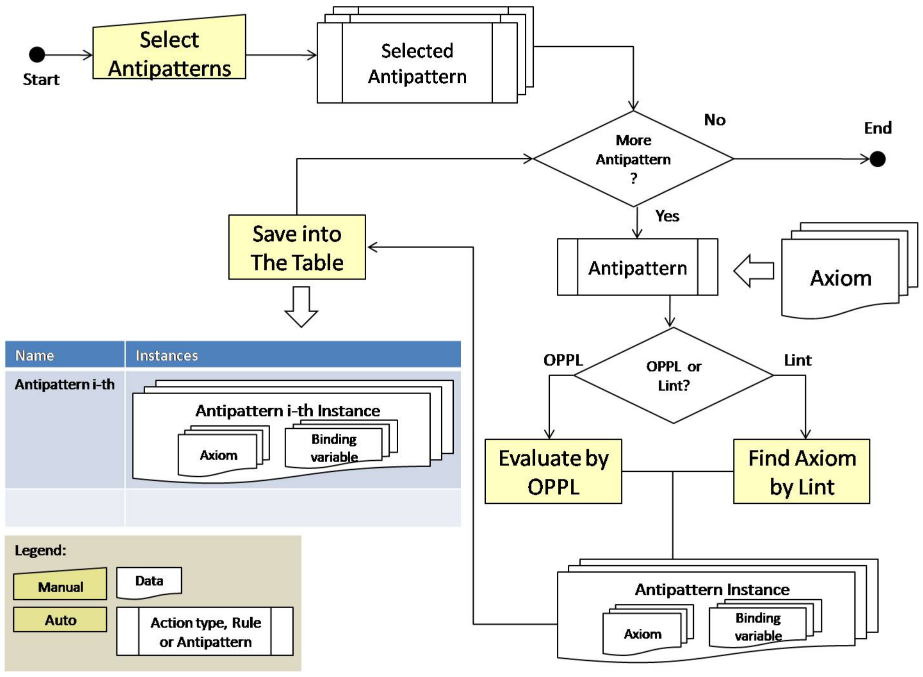


Figure 6.2: Detection process flowchart

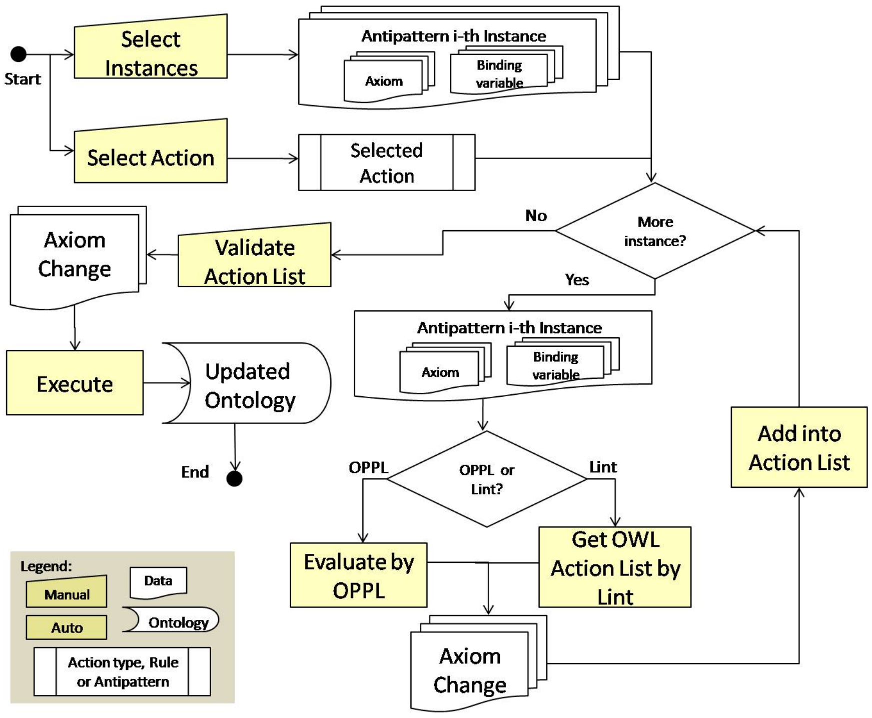


Figure 6.3: Execution process flowchart

of axiom changes in Apero. An axiom change is an action that has the ability to add or remove the mentioned axiom in an ontology.

The ontology developer can select more than one instance. Meanwhile, an antipattern has some possible actions. We will display their name in the list but he can only choose one. An action name is automatically set to the first option. Once he selects either instance or action, Apero automatically generates corrections as recommendations.

Like detection process, in this process, we apply two different implementation in order to generate a recommendation. On OPPL, we still use evaluation with additional binding variables on the detection part of a query.

A binding variable in OPPL syntax is a string ?[*variableName*]*MATCH* "[*variableValue*]". We put every binding variable sequentially at the end of the detection part. After Apero runs the evaluation, it will return a set of axiom change. On Lint implementation, the process runs easier. Apero will run a Java-class implementation of the action process with binding variables as input and generate a set of axiom changes. A binding variable in Lint implementation is just a vector or list of variable values. We do not need the variable name because the Java-class has already known it from the sequence in the vector. For every selected instance, Apero will collect all sets of axiom changes together in an action list.

This action list that contains a set of axiom change needs to be validated and contribution of users is very important here. They can delete unappropriated axiom changes or try another possible action. To finalize the process, Apero will execute all final axiom changes and affect directly into the current ontology. Since the ontology is updated, it is important to create new version of this ontology.

6.3 Implementation

6.3.1 Environment

The plug-in has been successfully built using Java programming language with the following environments:

- Java Development Kit 1.6 and Eclipse as the development tool.
- OWL API 2.0 under Protégé 4.0 to enable to use classes and functionality in Protégé.
- OPPL 2 API (org.coode.oppl-API.jar) to allow to access all OPPL function. [*]

[*]the source of API is available at *http://sourceforge.net/projects/oppl2/files/*

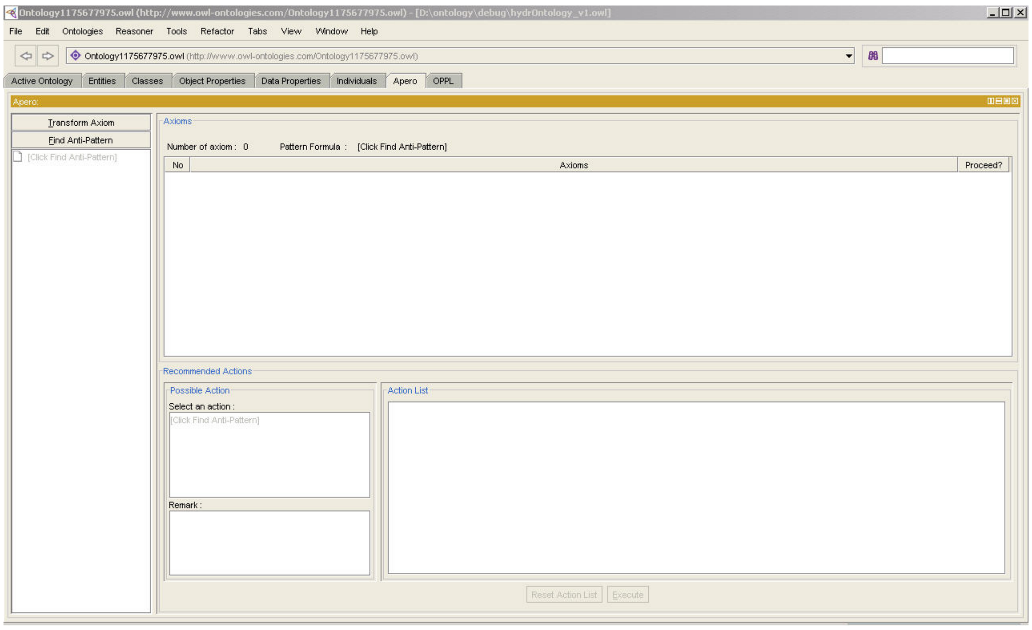

Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl) - [D:\ontology\debug\hydrOntology_v1.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals Apero OPPL
Apero:
Transform Axiom
Find Anti-Pattern
[Click Find Anti-Pattern]
Axioms
Number of axiom : 0
Pattern Formula : [Click Find Anti-Pattern]
No
Axioms
Proceed?
Recommended Actions
Possible Action
Select an action :
[Click Find Anti-Pattern]
Remark :
Action List
Reset Action List
Execute

Figure 6.4: Apero as a plug-in in Protégé

- JLatextMath 0.9.1 to give a display of formula as mathematic symbol in Latex. It will help users to understand how to describe an antipattern detection. [†]

The last two above has been bundled together with Apero implementation. The implementation of Apero itself is only a Jar (Java Archive) file that is executable only in Protégé. All compiled files in java classes and configuration file are archived in the Jar file that we name it as *org.upm.apero.jar*. In order to be recognized by Protégé, we have to put inside the folder $plug - ins$ in the Protégé installation folder. Apero has been tested on Windows system with Protégé 4.0 or above.

The figure 6.4 is appearance of Apero in Protégé at the first time Protégé loaded. We will find Apero in a window-tab of Protégé. Apero also can be accessed from the Protégé menu [Tabs >Apero] or [View >Ontology Views >Apero] (see figure 6.5 and 6.6).

6.3.2 Antipattern Implementation

Implementation of an antipattern is started from its configuration (see subsection 6.2.1). Configuration in Apero is provided as a java-properties file (see the Figure 6.7). Content of this file is a set of lines containing $[variableName] = [variableValue]$. Every non-optional parameter must be exist in this file.

[†]the source of JLatextMath 0.9.1 is available at

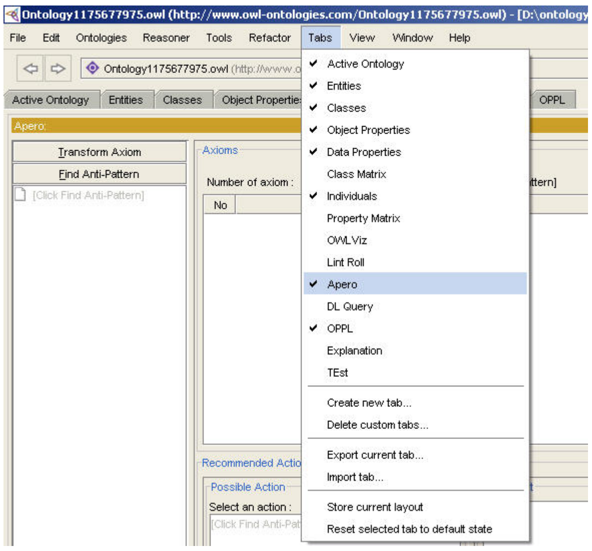


Figure 6.5: How to access Apero from menu [Tabs] in Protégé

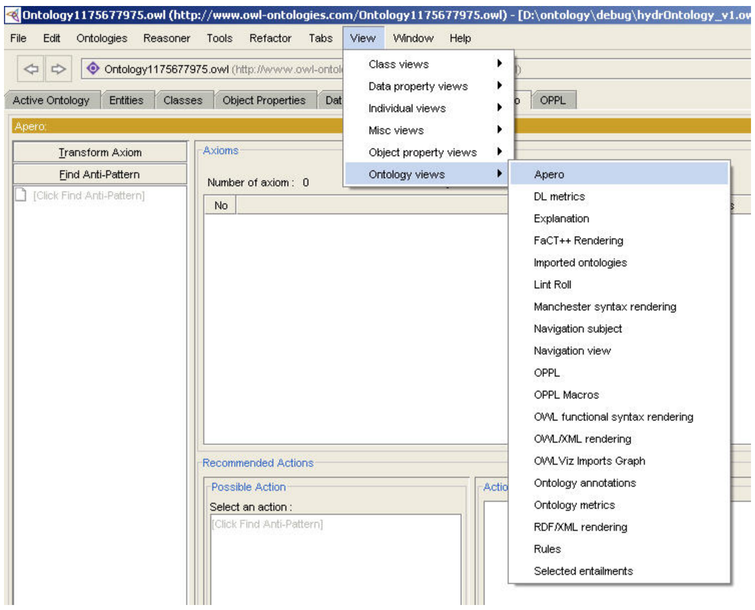
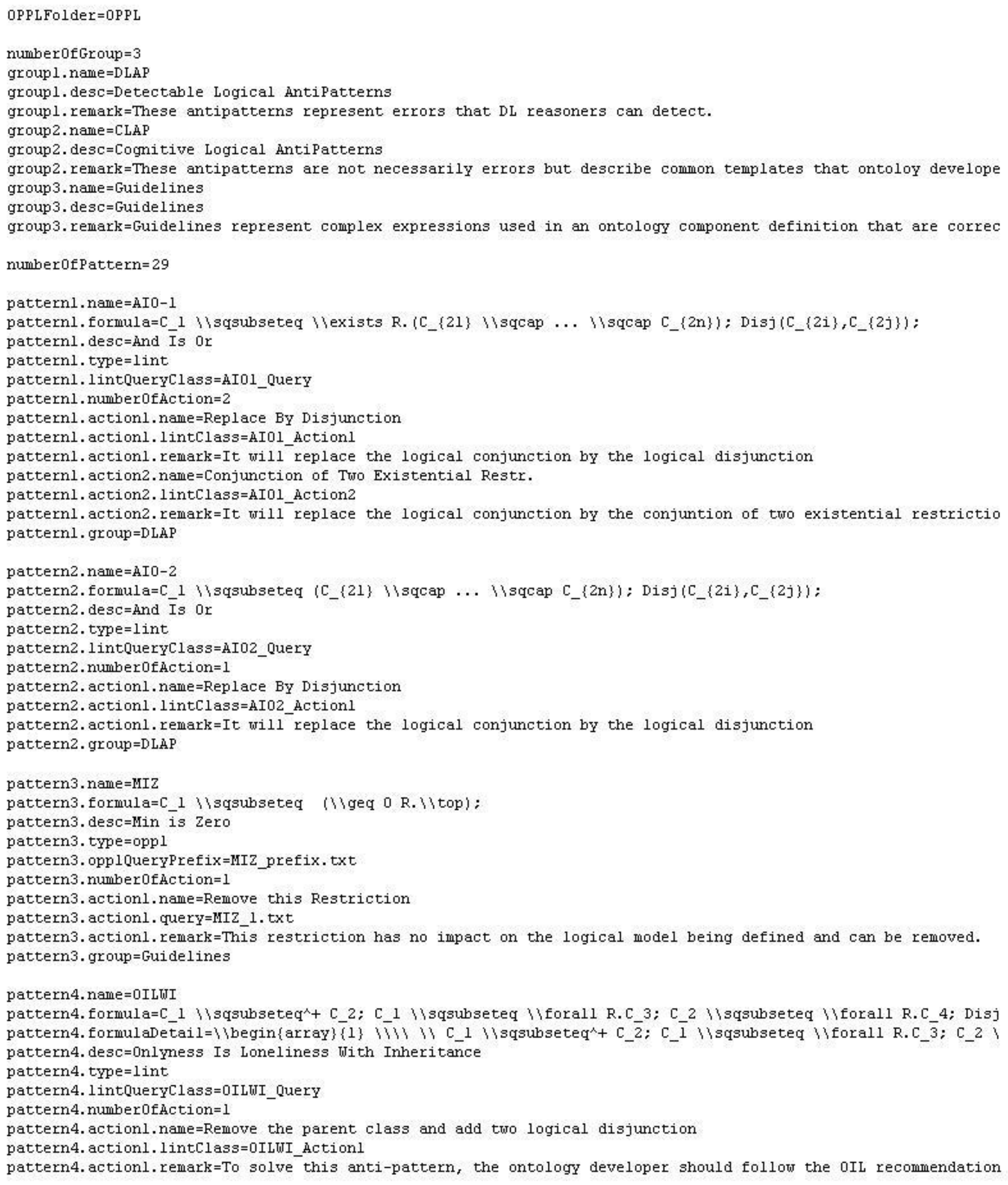


Figure 6.6: How to access Apero from menu [View] in Protégé



```
OPPLFolder=OPPL

numberOfGroup=3
group1.name=DLAP
group1.desc=Detectable Logical AntiPatterns
group1.remark=These antipatterns represent errors that DL reasoners can detect.
group2.name=CLAP
group2.desc=Cognitive Logical AntiPatterns
group2.remark=These antipatterns are not necessarily errors but describe common templates that ontoloy develope
group3.name=Guidelines
group3.desc=Guidelines
group3.remark=Guidelines represent complex expressions used in an ontology component definition that are correc

numberOfPattern=29

pattern1.name=AIO-1
pattern1.formula=C_1 \\sqsubseteq \\exists R.(C_{21} \\sqcap ... \\sqcap C_{2n}); Disj(C_{2i},C_{2j});
pattern1.desc=And Is Or
pattern1.type=lint
pattern1.lintQueryClass=AIO1_Query
pattern1.numberOfAction=2
pattern1.action1.name=Replace By Disjunction
pattern1.action1.lintClass=AIO1_Action1
pattern1.action1.remark=It will replace the logical conjunction by the logical disjunction
pattern1.action2.name=Conjunction of Two Existential Restr.
pattern1.action2.lintClass=AIO1_Action2
pattern1.action2.remark=It will replace the logical conjunction by the conjuntion of two existential restrictio
pattern1.group=DLAP

pattern2.name=AIO-2
pattern2.formula=C_1 \\sqsubseteq (C_{21} \\sqcap ... \\sqcap C_{2n}); Disj(C_{2i},C_{2j});
pattern2.desc=And Is Or
pattern2.type=lint
pattern2.lintQueryClass=AIO2_Query
pattern2.numberOfAction=1
pattern2.action1.name=Replace By Disjunction
pattern2.action1.lintClass=AIO2_Action1
pattern2.action1.remark=It will replace the logical conjunction by the logical disjunction
pattern2.group=DLAP

pattern3.name=MIZ
pattern3.formula=C_1 \\sqsubseteq  (\\geq 0 R.\\top);
pattern3.desc=Min is Zero
pattern3.type=oppl
pattern3.opplQueryPrefix=MIZ_prefix.txt
pattern3.numberOfAction=1
pattern3.action1.name=Remove this Restriction
pattern3.action1.query=MIZ_1.txt
pattern3.action1.remark=This restriction has no impact on the logical model being defined and can be removed.
pattern3.group=Guidelines

pattern4.name=OILWI
pattern4.formula=C_1 \\sqsubseteq^+ C_2; C_1 \\sqsubseteq \\forall R.C_3; C_2 \\sqsubseteq \\forall R.C_4; Disj
pattern4.formulaDetail=\\begin{array}{l} \\\\ \\ C_1 \\sqsubseteq^+ C_2; C_1 \\sqsubseteq \\forall R.C_3; C_2 \
pattern4.desc=Onlyness Is Loneliness With Inheritance
pattern4.type=lint
pattern4.lintQueryClass=OILWI_Query
pattern4.numberOfAction=1
pattern4.action1.name=Remove the parent class and add two logical disjunction
pattern4.action1.lintClass=OILWI_Action1
pattern4.action1.remark=To solve this anti-pattern, the ontology developer should follow the OIL recommendation
```

Figure 6.7: A Java-properties file

In previous chapters, an antipattern could have more than one detection type. For instance, AIO has two detection types (see 5.1 and 5.2). We consider it in Apero as two different antipatterns. Hence, we use numbering on their name to keep the original antipattern. Our research has defined three types of antipattern, thus we name them in Apero as DLAP, CLAP and Guideline.

The current implementation has 29 antipatterns consisting 4 OPPL and 25 Lint-based antipatterns, distributed with 20 antipatterns in DLAP, 2 antipatterns in CLAP and 7 antipatterns in Guidelines. An antipattern is configured according to our design in Table 6.2 and 6.3. In the file, we found it with additional prefix $pattern[i]$ where [i] is a sequence number. We use dot symbol to separate antipattern sequence and its parameter. We also use this for the group of antipattern.

In the Figure 6.7, we have MIZ as example of OPPL-based antipattern. The Code 45 and 46 are content of file $MIZ\_prefix.txt$ and $MIZ\_1.txt$, respectively. Those file must be exist in the folder $OPPLFolder$ parameter which is $OPPL$. The first file exactly represents the formula of MIZ (see 4.5) and $formula$ parameter, while the second file represents the recommendation formula (see 4.6).

```
?c1:CLASS,
?r:OBJECTPROPERTY
Select
   ?c1 subClassOf ?r min 0 Thing
Where ?c1 != Thing
```

Code 45: File MIZ_prefix.txt

```
begin
   remove ?c1 subClassOf ?r min 0 Thing
end;
```

Code 46: File MIZ_1.txt

A Lint-based antipattern implementation is programmable with Java programming language. The figure 6.8 displays a UML class diagram (without stereotype [‡]) of the implementation. We provide OILWI antipattern as an example. According to the configuration, OILWI has a detection class $OILWI\_Query$

http://forge.scilab.org/index.php/p/jlatexmath/downloads/

[‡]Graphically, a stereotype is rendered as a name enclosed by ≪≫ and placed above the name of another element. In addition or alternatively it may be indicated by a specific icon.

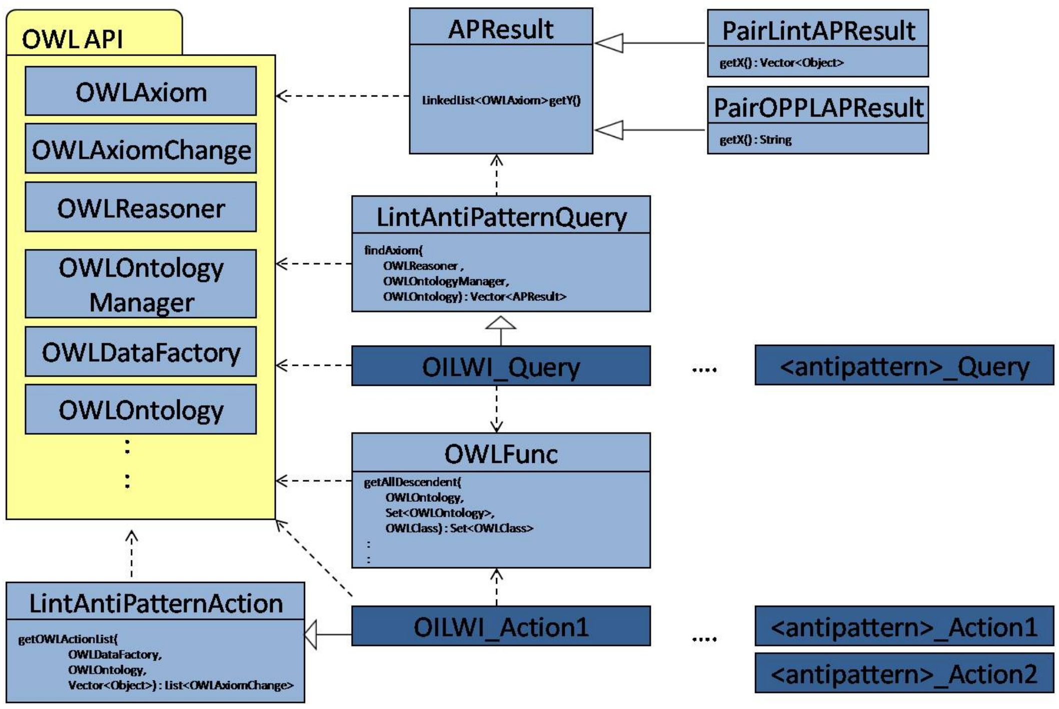


Figure 6.8: A UML Class Diagram Lint-based Antipattern implementation

and an action class $OILWI\_Action1$. If an antipattern has more than one possible action, it also has more than one action class. The class name is arbitrary. Two antipatterns probably share the same action, then they can use the same class.

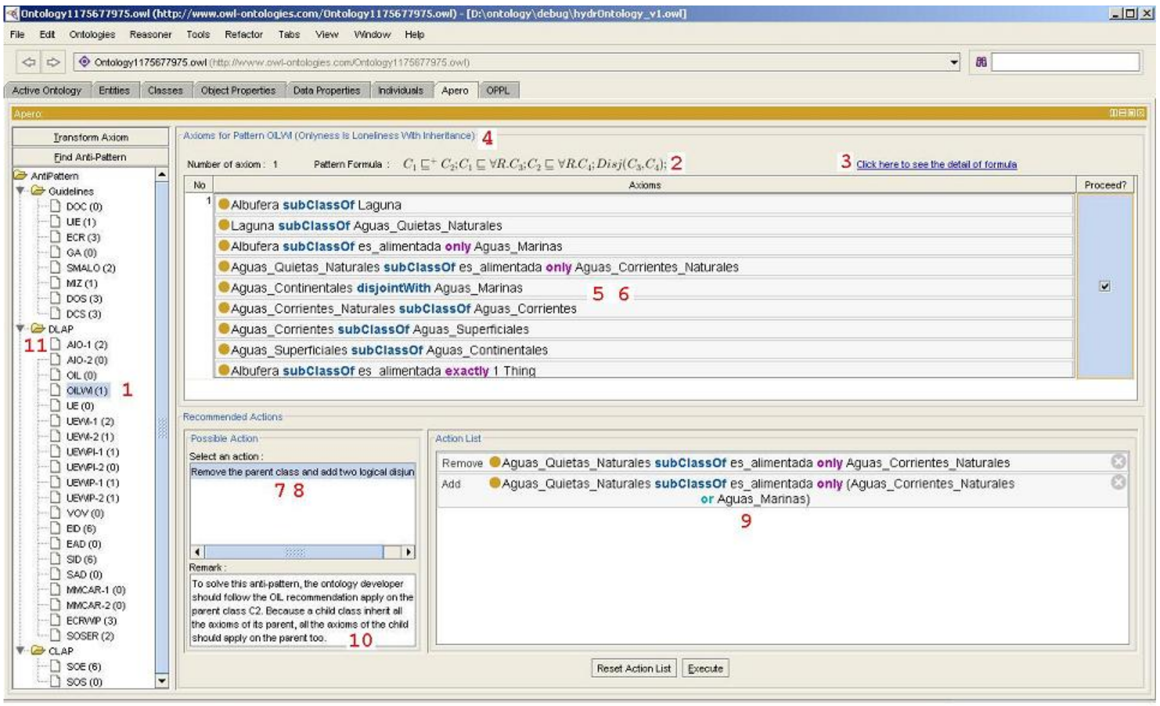
A detection class must implement the following methods : [§]

- $findAxiom(OWLReasoner, OWLOntologyManager, OWLOntology)$ from interface $LintAntiPatternQuery$.
  $OWLReasoner$ is a reasoner implementation that may be used by a developer for a certain purpose. As an exchange of using reasoner, additional response time may be applied. $OWLOntologyManager$ is a class that enables the developer to manipulate an ontology. $OWLOntology$ represents the current ontology.

- $getOWLActionList(OWLDataFactory, OWLOntology, Vector < Object >$ $)$ from interface $LintAntiPatternAction$.
  The purpose of using $OWLDataFactory$ is to create new axioms and $Vector < Object >$ is a representation of binding variables.

Both methods represent implementation of the box $findAxiomByLint$ and $GetOWLActionListByLint$ on the figure 6.2 and 6.3, respectively. Vector of

[§]This is a term in Java programming language that means a procedure or function.



Axioms for Pattern OILWI (Onlyness Is Loneliness With Inheritance) 4

Number of axiom : 1 Pattern Formula : $C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; C_2 \sqsubseteq \forall R.C_4; Disj(C_3,C_4);$ 2

3 Click here to see the detail of formula

No	Axioms	Proceed?
1	● Albufera **subClassOf** Laguna ● Laguna **subClassOf** Aguas_Quietas_Naturales ● Albufera **subClassOf** es_alimentada **only** Aguas_Marinas ● Aguas_Quietas_Naturales **subClassOf** es_alimentada **only** Aguas_Corrientes_Naturales ● Aguas_Continentales **disjointWith** Aguas_Marinas 5 6 ● Aguas_Corrientes_Naturales **subClassOf** Aguas_Corrientes ● Aguas_Corrientes **subClassOf** Aguas_Superficiales ● Aguas_Superficiales **subClassOf** Aguas_Continentales ● Albufera **subClassOf** es_alimentada **exactly** 1 Thing	✓

Recommended Actions

Possible Action

Select an action :

Remove the parent class and add two logical disjun

7 8

Remark :

To solve this anti-pattern, the ontology developer should follow the OIL recommendation apply on the parent class C2. Because a child class inherit all the axioms of its parent, all the axioms of the child should apply on the parent too. 10

Action List

Remove ● Aguas_Quietas_Naturales **subClassOf** es_alimentada **only** Aguas_Corrientes_Naturales

Add ● Aguas_Quietas_Naturales **subClassOf** es_alimentada **only** (Aguas_Corrientes_Naturales **or** Aguas_Marinas)

9

```
1  pattern4.name=OILWI
2  pattern4.formula=C_1 \\sqsubseteq^+ C_2; C_1 \\sqsubseteq \\forall R.C_3; C_2 \\sqsubseteq \\forall R.C_4; Disj(
3  pattern4.formulaDetail=\\begin{array}{l} \\\\ \\ C_1 \\sqsubseteq^+ C_2; C_1 \\sqsubseteq \\forall R.C_3; C_2 \\
4  pattern4.desc=Onlyness Is Loneliness With Inheritance
5  pattern4.type=lint
6  pattern4.lintQueryClass=OILWI_Query
7  pattern4.numberOfAction=1
8  pattern4.action1.name=Remove the parent class and add two logical disjunction
9  pattern4.action1.lintClass=OILWI_Action1
10 pattern4.action1.remark=To solve this anti-pattern, the ontology developer should follow the OIL recommendation
11 pattern4.group=DLAP
```

Figure 6.9: A mapping antipattern configuration to the GUI

class *APResult* is output of detection class. Class *APResult* has the matched axioms. If we proceed a OPPL-based antipattern, then we will get instances of class *PairOPPLAPResult* (sub class of *APResult*) that also has binding variables completing the matched axioms as a pair. Meanwhile, processing a Lint-based antipattern, we will get instance of class *PairLintAPResult* (sub class of *APResult*) that also has binding variables namely vector of variable value. The different between class *PairOPPLAPResult* and *PairLintAPResult* is the representation of binding variables like we discuss in design section.

In order to help a developer of antipattern, we provide a class *OWLFunc* that has a lot of functions to solve some problem, for instance to get all subclasses of a class and get all axioms performing disjointness between two classes. You find the complete list of function in the Appendix A.

The figure 6.9 shows how every parameter in configuration displayed in the Apero. We have several remarks as follows:

- Parameter number 1, 4 and 10 will be displayed directly at the position shown in the figure.

- Apero will render parameter number 2 becoming latex symbol
- Parameter number 3 yields a hyperlink. If the ontology developer clicks on it, a new window will appear displaying latex formula with a bigger area than parameter number two. Parameter number 3 does not exist then that link will not exist either.
- Parameter number 5 and 6 will give output and put it into the table as shown on the figure, no matter what type of implementation.
- Parameter number 7 remunerates number of action and each name of action (parameter number 8) will be displayed.
- Parameter number 9 gives output and put it into the list.
- Parameter number 11 will categorize this antipattern and perform a tree.

6.3.3 Transformation Process Implementation

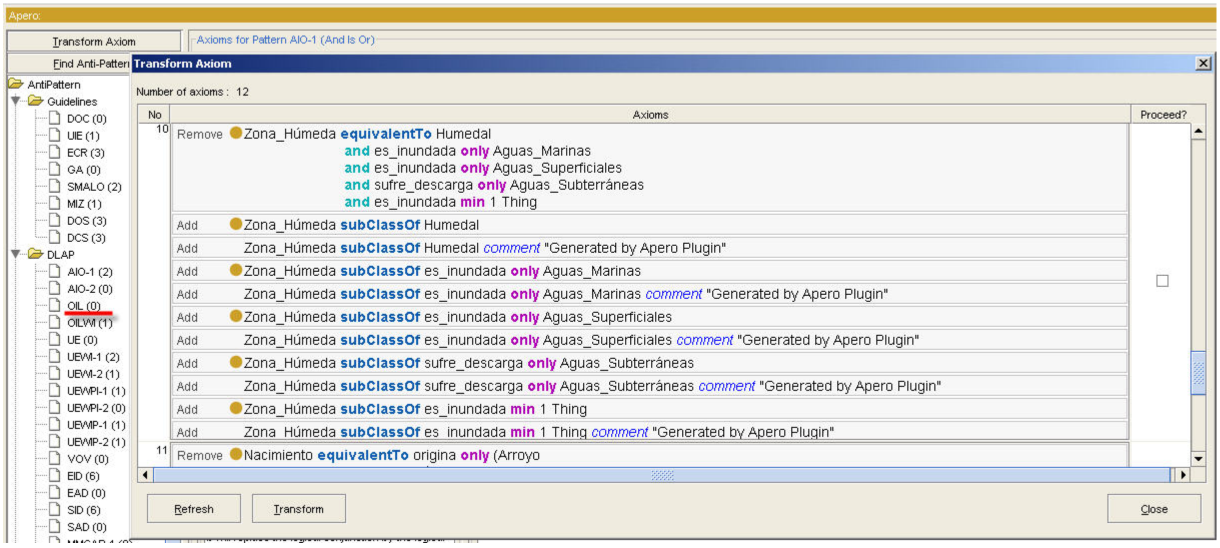
This process is very important because it is potential to emerge an antipattern. The figure 6.10 is the interface for user to enable the transformation. We have the OIL antipattern as an example here. Before transformation, we do not have OIL antipattern. The ontology developer starts this transformation by pushing the button [Transform Axiom]. Apero will display all possible axiom transformation in a table. Every row in the table consists of a number, list of axiom completed with its actions, and a check box [Proceed] that indicates whether you want to transform this row or not. An original axiom is begun with an action $REMOVE$, while a result axiom is begun with an action ADD.

A new annotation comment axiom "Generated by Apero Plug-in" appears to indicate this axiom came from the transformation process as an acknowledgment for the ontology developer. The figure 6.11 shows the new axioms with different icon of annotation since they have an annotation.

After transformation, Apero successfully detects the OIL antipattern (see figure 6.12). Clearly, three axioms from the instance of antipattern come from the equivalent axiom before transformation process. Finally, it is necessary to create new version of ontology since both ontology before and after transformation are not the same.

6.3.4 Detection Process Implementation

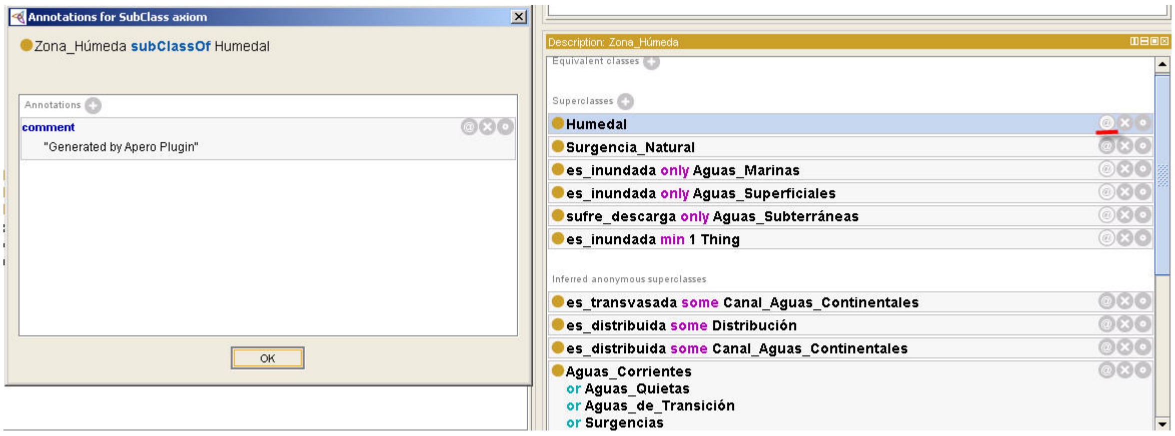
The figure 6.13 tells how the ontology developer runs a detection process. He has to click button [Find Antipattern] to display dialog [Find Antipattern]. On this dialog, he can select some antipatterns that he wants to observe. [Ellipse time] indicates the processing time to run the process. If he only wants to display the



Number of axioms : 12

No		Axioms	Proceed?
10	Remove	● Zona_Húmeda **equivalentTo** Humedal **and** es_inundada **only** Aguas_Marinas **and** es_inundada **only** Aguas_Superficiales **and** sufre_descarga **only** Aguas_Subterráneas **and** es_inundada **min** 1 Thing	☐
	Add	● Zona_Húmeda **subClassOf** Humedal	
	Add	Zona_Húmeda **subClassOf** Humedal *comment* "Generated by Apero Plugin"	
	Add	● Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Marinas	
	Add	Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Marinas *comment* "Generated by Apero Plugin"	
	Add	● Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Superficiales	
	Add	Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Superficiales *comment* "Generated by Apero Plugin"	
	Add	● Zona_Húmeda **subClassOf** sufre_descarga **only** Aguas_Subterráneas	
	Add	Zona_Húmeda **subClassOf** sufre_descarga **only** Aguas_Subterráneas *comment* "Generated by Apero Plugin"	
	Add	● Zona_Húmeda **subClassOf** es_inundada **min** 1 Thing	
	Add	Zona_Húmeda **subClassOf** es_inundada **min** 1 Thing *comment* "Generated by Apero Plugin"	
11	Remove	● Nacimiento **equivalentTo** origina **only** (Arroyo	

Figure 6.10: Before transformation



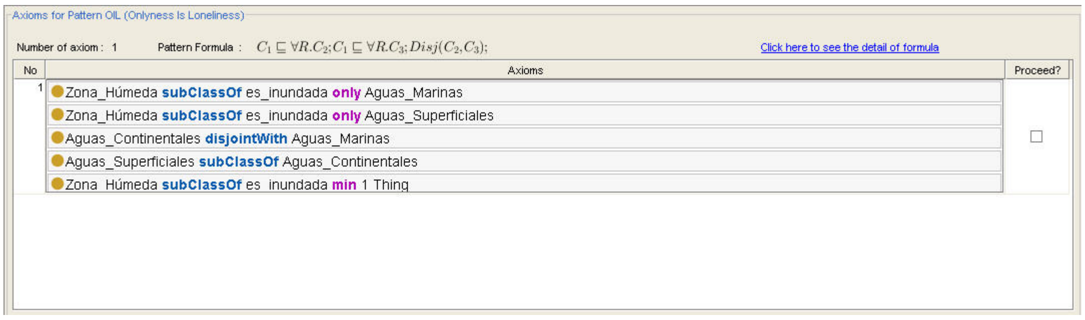
● Zona_Húmeda **subClassOf** Humedal

comment
"Generated by Apero Plugin"

● **Humedal**
● **Surgencia_Natural**
● **es_inundada only Aguas_Marinas**
● **es_inundada only Aguas_Superficiales**
● **sufre_descarga only Aguas_Subterráneas**
● **es_inundada min 1 Thing**

● **es_transvasada some Canal_Aguas_Continentales**
● **es_distribuida some Distribución**
● **es_distribuida some Canal_Aguas_Continentales**
● **Aguas_Corrientes or Aguas_Quietas or Aguas_de_Transición or Surgencias**

Figure 6.11: An annotation acknowledgment


Axioms for Pattern OIL (Onlyness Is Loneliness)
Click here to see the detail of formula

Number of axiom : 1 Pattern Formula : $C_1 \sqsubseteq \forall R.C_2; C_1 \sqsubseteq \forall R.C_3; Disj(C_2, C_3);$

No	Axioms	Proceed?
1	● Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Marinas	☐
	● Zona_Húmeda **subClassOf** es_inundada **only** Aguas_Superficiales	
	● Aguas_Continentales **disjointWith** Aguas_Marinas	
	● Aguas_Superficiales **subClassOf** Aguas_Continentales	
	● Zona_Húmeda **subClassOf** es_inundada **min** 1 Thing	

Figure 6.12: After transformation

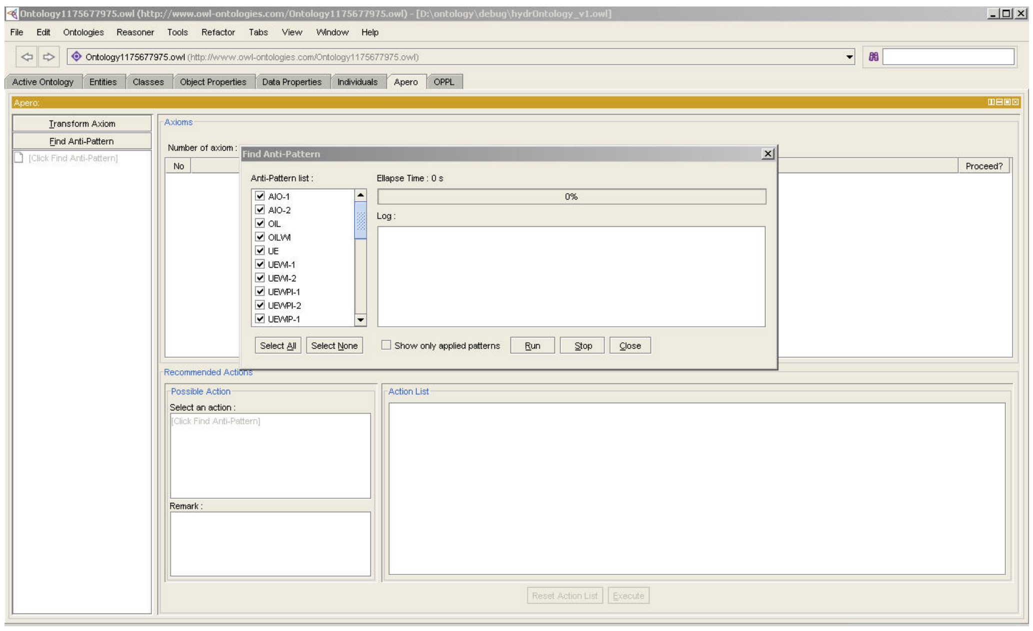

Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl) - [D:\ontology\debug\hydrOntology_v1.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Ontology1175677975.owl (http://www.owl-ontologies.com/Ontology1175677975.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals Apero OPPL
Apero:
Transform Axiom
Find Anti-Pattern
[Click Find Anti-Pattern]
Axioms
Number of axiom :
No
Proceed?
Find Anti-Pattern
Anti-Pattern list :
AIO-1
AIO-2
OIL
OILWI
UE
UEWI-1
UEWI-2
UEWPI-1
UEWPI-2
UEWIP-1
Ellapse Time : 0 s
0%
Log :
Select All
Select None
Show only applied patterns
Run
Stop
Close
Recommended Actions
Possible Action
Select an action :
[Click Find Anti-Pattern]
Remark :
Action List
Reset Action List
Execute

Figure 6.13: Before detection process

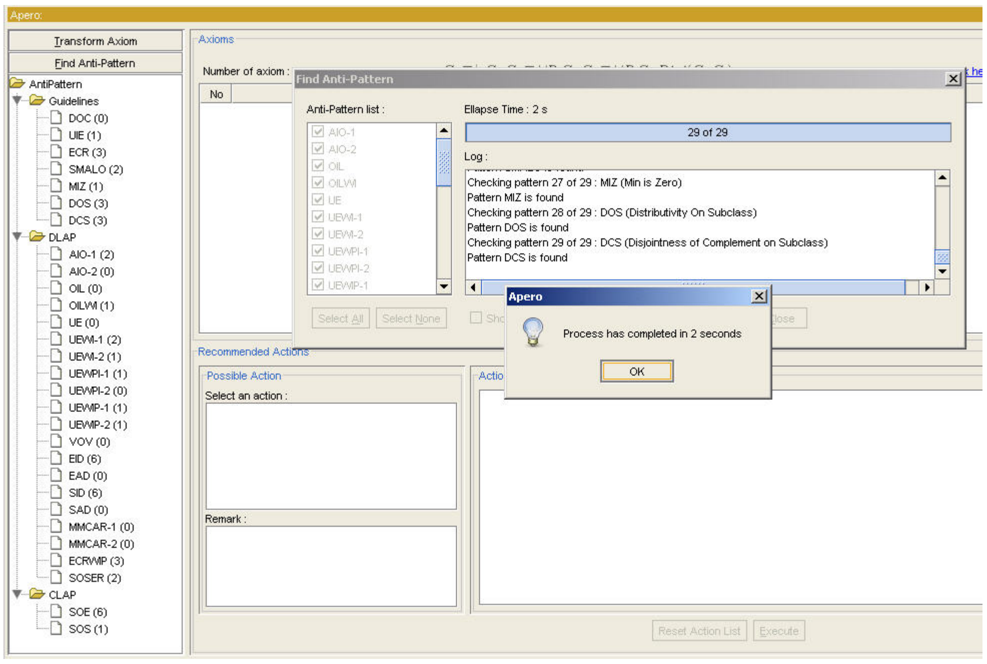
applied antipattern, he has to click check box [Show Only applied patterns]. Text area [Log] will record all activity list of this process. Button [Run] will start the detection process, while button [Stop] will stop detecting.

After finishing detection, the dialog will show a message completeness and Apero will update the tree of antipatterns at the left side of the window (see figure 6.14). Every antipattern name will be displayed together with the number of instances found in the ontology.

The ontology developer may want to observe an antipattern, for example : SMALO antipattern. He can click on an antipattern in the tree of antipattern. As the result, Apero will display all instances of SMALO antipattern in this ontology. Apero also will display all available information of the antipattern configuration. If we crosscheck every instance to pattern formula, they should match each other with a certain substitution of variables.

6.3.5 Execution Process Implementation

The figure 6.16 and 6.17 describe an execution process. We use AIO as an example here. After observation, the ontology developer realizes that there are something wrong in the ontology. Two instances of AIO antipattern have proved it. A recommendation is built once he selects an instance to be processed by marking on the check box [proceed] column in the table. He may want to see the compound recommendation by selecting more than one instance. A possible action will trigger a recommendation (see the figure 6.17). A description of the

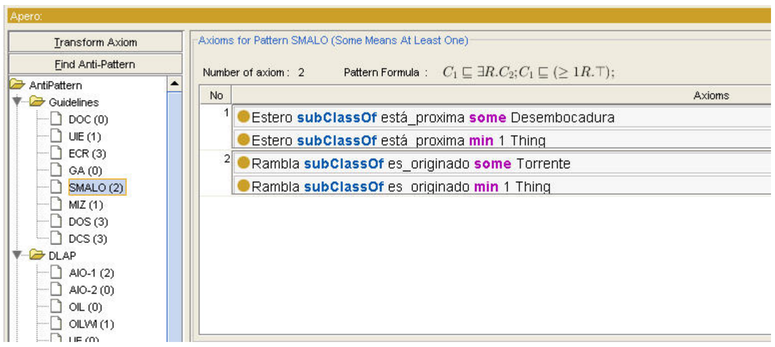


```
Checking pattern 27 of 29 : MIZ (Min is Zero)
Pattern MIZ is found
Checking pattern 28 of 29 : DOS (Distributivity On Subclass)
Pattern DOS is found
Checking pattern 29 of 29 : DCS (Disjointness of Complement on Subclass)
Pattern DCS is found
```

Apero

Process has completed in 2 seconds

Figure 6.14: After detection process

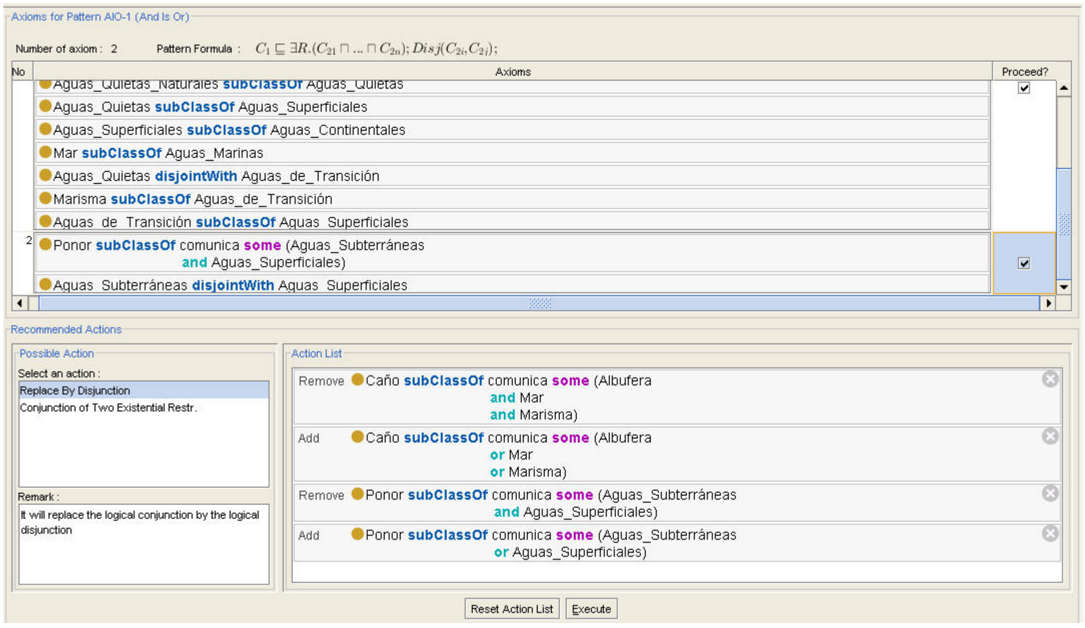


Axioms for Pattern SMALO (Some Means At Least One)

Number of axiom : 2 Pattern Formula : $C_1 \sqsubseteq \exists R.C_2; C_1 \sqsubseteq (\geq 1 R.\top);$

No	Axioms
1	● Estero **subClassOf** está_proxima **some** Desembocadura
	● Estero **subClassOf** está_proxima **min** 1 Thing
2	● Rambla **subClassOf** es_originado **some** Torrente
	● Rambla **subClassOf** es_originado **min** 1 Thing

Figure 6.15: SMALO antipattern


Axioms for Pattern AIO-1 (And Is Or)

Number of axiom : 2 Pattern Formula : $C_1 \sqsubseteq \exists R.(C_{21} \sqcap ... \sqcap C_{2n}); Disj(C_{2i}, C_{2j});$

No	Axioms	Proceed?
	Aguas_Quietas_Naturales subClassOf Aguas_Quietas	✓
	Aguas_Quietas subClassOf Aguas_Superficiales	
	Aguas_Superficiales subClassOf Aguas_Continentales	
	Mar subClassOf Aguas_Marinas	
	Aguas_Quietas disjointWith Aguas_de_Transición	
	Marisma subClassOf Aguas_de_Transición	
	Aguas_de_Transición subClassOf Aguas_Superficiales	
2	Ponor subClassOf comunica some (Aguas_Subterráneas and Aguas_Superficiales)	✓
	Aguas_Subterráneas disjointWith Aguas_Superficiales	

Recommended Actions

Possible Action

Select an action :

Replace By Disjunction

Conjunction of Two Existential Restr.

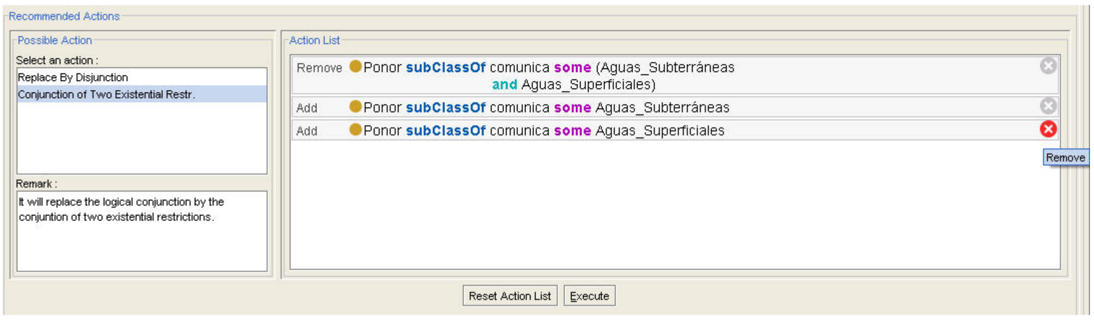
Remark :

It will replace the logical conjunction by the logical disjunction

Action List

```
Remove  Caño subClassOf comunica some (Albufera
                                        and Mar
                                        and Marisma)
Add     Caño subClassOf comunica some (Albufera
                                        or Mar
                                        or Marisma)
Remove  Ponor subClassOf comunica some (Aguas_Subterráneas
                                         and Aguas_Superficiales)
Add     Ponor subClassOf comunica some (Aguas_Subterráneas
                                         or Aguas_Superficiales)
```

Figure 6.16: Execution process 1


Recommended Actions

Possible Action

Select an action :

Replace By Disjunction

Conjunction of Two Existential Restr.

Remark :

It will replace the logical conjunction by the conjuntion of two existential restrictions.

Action List

```
Remove  Ponor subClassOf comunica some (Aguas_Subterráneas
                                         and Aguas_Superficiales)
Add     Ponor subClassOf comunica some Aguas_Subterráneas
Add     Ponor subClassOf comunica some Aguas_Superficiales
```

Figure 6.17: Execution process 2

chosen action is given by the text [Remark].

A recommendation is composed by an action list that contains the list of axiom changes. The ontology developer needs to validate every axiom change with the domain expert. Probably, he may need to remove an axiom change. He also can reset the recommendation to the initial condition by using the button [Reset]. After he is sure, he can push the button [execute] that will execute all axiom change and permanently update the current ontology. The effect of execution on the example of AIO is described on the figure 6.18 and 6.19. The final step, he need to create new version of ontology because of this execution.

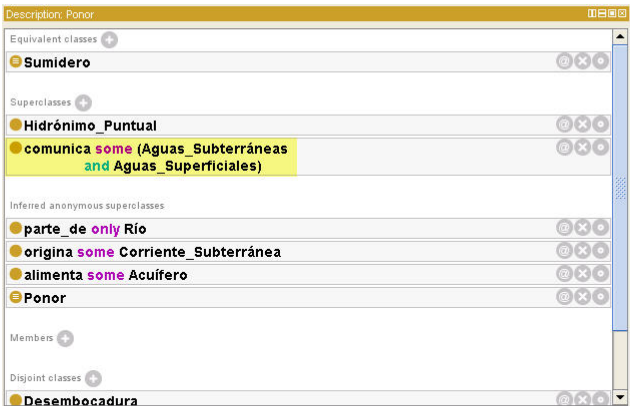

Description: Ponor
Equivalent classes
Sumidero
Superclasses
Hidrónimo_Puntual
comunica some (Aguas_Subterráneas and Aguas_Superficiales)
Inferred anonymous superclasses
parte_de only Río
origina some Corriente_Subterránea
alimenta some Acuífero
Ponor
Members
Disjoint classes
Desembocadura

Figure 6.18: Description of class *Ponor* before execution

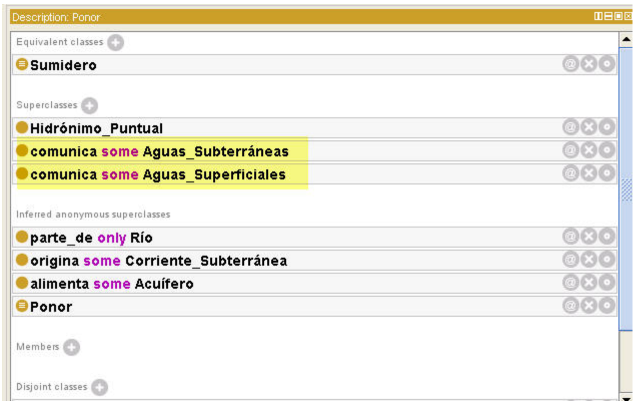

Description: Ponor
Equivalent classes
Sumidero
Superclasses
Hidrónimo_Puntual
comunica some Aguas_Subterráneas
comunica some Aguas_Superficiales
Inferred anonymous superclasses
parte_de only Río
origina some Corriente_Subterránea
alimenta some Acuífero
Ponor
Members
Disjoint classes

Figure 6.19: Description of class *Ponor* after execution

6.4 Debugging Strategy Based on Antipatterns

An initial study about debugging strategy based on antipatterns has been established in [4] as shown in the Figure 6.20. All antipattern appeared in the figure, are the list of antipattern in [4].

New antipatterns have been discovered and a debugging strategy must be revised. Users can use Apero plug-in to apply the strategy easily and the strategy itself will guide users how to debug an ontology optimally. Antipattern may lead to another antipattern after executing a recommendation. It triggers a dependency among antipattern in the strategy.

The figure 6.21 displays a new debugging strategy six steps. The first step of strategy follows the one in the former debugging strategy. Applying SOE will affect removing a class. It means that our debugging will be easier since if the class is unsatisfiable then the number of unsatisfiable class decreased by one. The second step is to check Guidelines that use semantic equivalences between formulas in a recommendation. They are the DCS and DOS antipatterns. Applying these

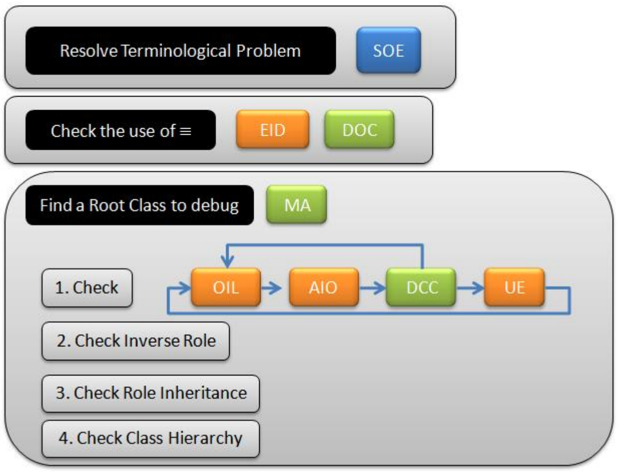


Figure 6.20: Debugging Strategy in [4]

antipatterns will give positive effect of antipattern finding for the next step. The third step follows the second of the old strategy but, there are new two additional antipattern namely EAD and UIE antipattern.

At the fourth step, there are the rest of DLAP, SOS and ECR. The sequence in this step is optional. All antipattern can be combined as presented at the figure. A new instance of antipattern is possible to be detected after several combinations. At some point it will stop and you may continue to the next step. Users can take freedom of DLAP antipatterns as stopping criteria because the existence of DLAP implies there is at least one unsatisfiable class. The fifth step is to remove superfluous axioms possibly detected by SMALO and MIZ antipattern. We keep this axiom on the previous step because of performance reason. Some antipatterns (OIL and OILWI) in DLAP need minimality property that can be supported by one of axioms detected by antipatterns in this step.

At the sixth step, users probably need to know whether the ontology has already been free of unsatisfiable class by classifying ontology. If there is no unsatisfiable class, debugging process is done. Otherwise, users need to check on the transformation dialog whether there is a suggestion or not. If there is a suggestion, probably users need to clarify whether their modeling is correct. Users may decide to stop if DLAP category is no longer detected with one note which is a potential error or inconsistency that may appear in the future.

If reasoner detects an unsatisfiable class but Apero detects no DLAP antipattern, then new antipattern must be discovered. The ideal condition is all antipatterns in the world have been discovered. Surely, participation of reasoner to classify the ontology is no longer needed.

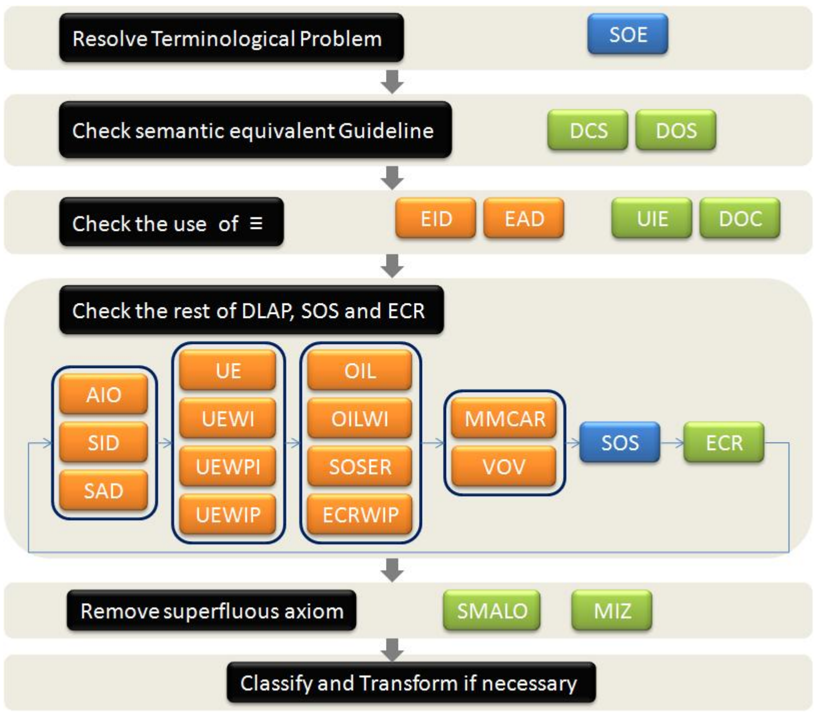

Resolve Terminological Problem
SOE
Check semantic equivalent Guideline
DCS
DOS
Check the use of ≡
EID
EAD
UIE
DOC
Check the rest of DLAP, SOS and ECR
AIO
SID
SAD
UE
UEWI
UEWPI
UEWIP
OIL
OILWI
SOSER
ECRWIP
MMCAR
VOV
SOS
ECR
Remove superfluous axiom
SMALO
MIZ
Classify and Transform if necessary

Figure 6.21: New Debugging Strategy

Chapter 7

Evaluation

This chapter presents the evaluation of the Apero plug-in. The evaluation shows that the Apero plug-in works as expected helping ontology developers to debug an ontology.

7.1 Evalution Setup

The evaluation is done in a computer with the following hardware and software:

- CPU : Pentium Dual Core T250 2.00Hz.
- RAM : 904MB
- Storage : 18GB
- Microsoft Windows Server 2003 SP 2
- Java : Java Development Kit 1.6
- Protege : Protege 4.0
- SWOOP 2.3 as debugging tool for comparison purpose

7.2 Evalution Test Case and Plan

We prepare some ontologies as test cases to help us evaluating the Apero plug-in. We also design them in a way so that they confirm with the work objectives defined in the chapter 3. The table 7.1 displays the list of ontology used for our evaluation.

We use a simple scenario to run test cases according to the figure 6.1 that represents a global strategy for ontology debugging. Classifying an ontology by

Ontology name	Domain	Ontology references	Languages	Total number of classes	Number of unsatisfiable classes
Computer_Science	University organisation	[1]	english	29	9
Tambis_full	bioinformatic	[19]	english	395	144
Sweet_Numeric	Earth and Environment	[17]	english	2364	2
HydrOntology	hydrology	[22]	spanish	159	114

Table 7.1: List of ontologies

reasoner before running Apero is not needed, except if users want to highlight some classes that are known unsatisfiable. However, users must remember that classifying a big and complex ontology is a time-consuming task. Therefore, we may skip two steps on the figure, namely [choose root of unsatisfiable class] and [compute justification and inspect class definition].

As part of the scenario, we need to check if new formula representation will be effective to show an antipattern. We need to remember tha a given latex formula does not drive an instance of antipattern in the implementation directly. In other words, it only gives visual representation of an antipattern with its instance to users.

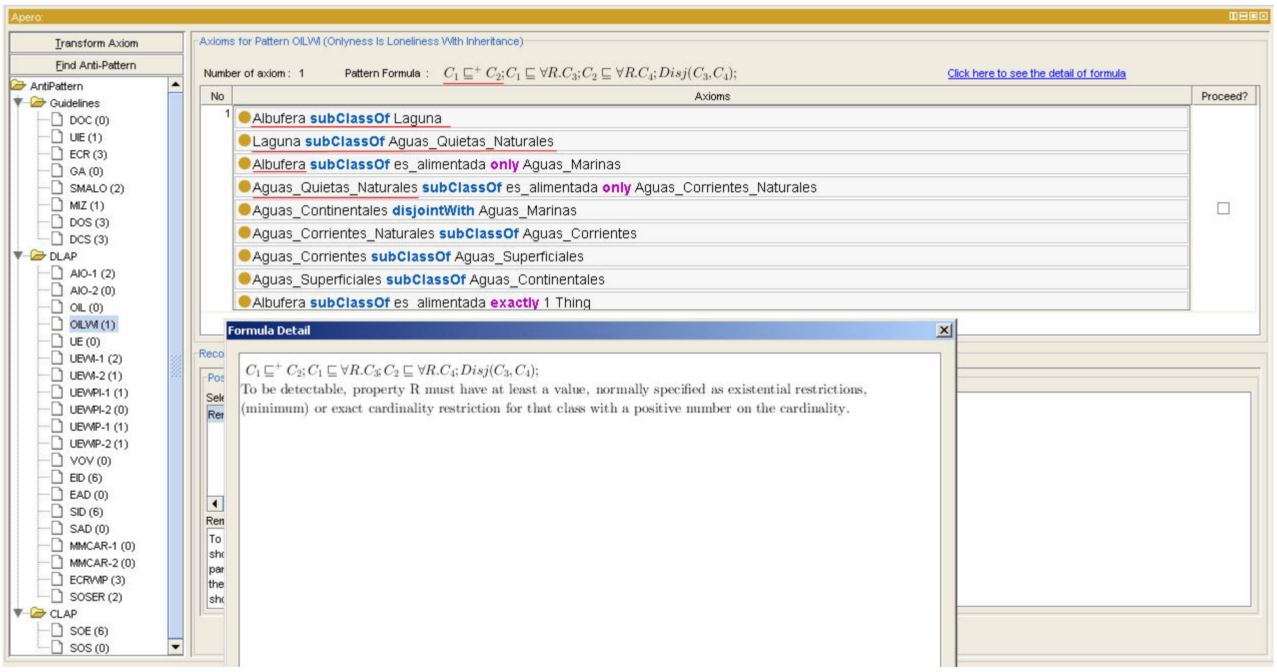
At the end of this evaluation, we expect to get some measurement about response time and detection result. In order to measure response time of each debugging, we run test for each ontology three times and take the average as the result. While measuring response time and detecting antipatterns, we only take the first cycle of detection without applying transformation rule, although the transformation rule may lead to more findings. In addition, we apply all antipatterns listed in this thesis.

Comparison with another debugging tool will show pros and cons of Apero. We will compare Apero to SWOOP. All test cases above will participate in this comparison.

7.3 Evaluation Result

The figure 7.1 shows new representation representing better than the old one. In this example, we use symbol + and without this symbol, it is difficult to explain to users how the formula is connected to the instance of antipattern. We give underline on some classes to show $C_1 \sqsubseteq^+ C_2$ representing $Albufera \sqsubseteq Laguna; Laguna \sqsubseteq Aguas\_Quietas\_Naturales$ with substitution :

- $C_1 := Albufera$



Axioms for Pattern OILWI (Onlyness Is Loneliness With Inheritance)

Number of axiom : 1 Pattern Formula : $C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; C_2 \sqsubseteq \forall R.C_4; Disj(C_3, C_4);$

No	Axioms	Proceed?
1	● Albufera **subClassOf** Laguna ● Laguna **subClassOf** Aguas_Quietas_Naturales ● Albufera **subClassOf** es_alimentada **only** Aguas_Marinas ● Aguas_Quietas_Naturales **subClassOf** es_alimentada **only** Aguas_Corrientes_Naturales ● Aguas_Continentales **disjointWith** Aguas_Marinas ● Aguas_Corrientes_Naturales **subClassOf** Aguas_Corrientes ● Aguas_Corrientes **subClassOf** Aguas_Superficiales ● Aguas_Superficiales **subClassOf** Aguas_Continentales ● Albufera **subClassOf** es alimentada **exactly** 1 Thing	☐

$C_1 \sqsubseteq^+ C_2; C_1 \sqsubseteq \forall R.C_3; C_2 \sqsubseteq \forall R.C_4; Disj(C_3, C_4);$
To be detectable, property R must have at least a value, normally specified as existential restrictions, (minimum) or exact cardinality restriction for that class with a positive number on the cardinality.

Figure 7.1: New formula representation

- $C_2 := Aguas\_Quietas\_Naturales$

In addition, the window [Formula detail] that appears after a user clicks the hyperlink [Click here to see the detail of formula] will give users extra space to express any condition more flexible.

The Table 7.2 is a summary of evaluation following the given scenario in 7.2. The table shows that all test cases are done in less than 30 seconds even for big ontology Sweet_Numeric. HydrOntology, the complex among test cases, is done only 3 seconds and we find 18 antipatterns with 45 instances.

Ontology	Response time	Total		Dominant	
	(second)	Antipattern	Instance	Antipattern	Instance
Computer_Science	<1	3	4	UEWIP	2
Tambis_full	1	5	42	DOS	27
Sweet_Numeric	27	4	10	SOE	4
HydrOntology	3	18	45	EID,SID,SOE	6

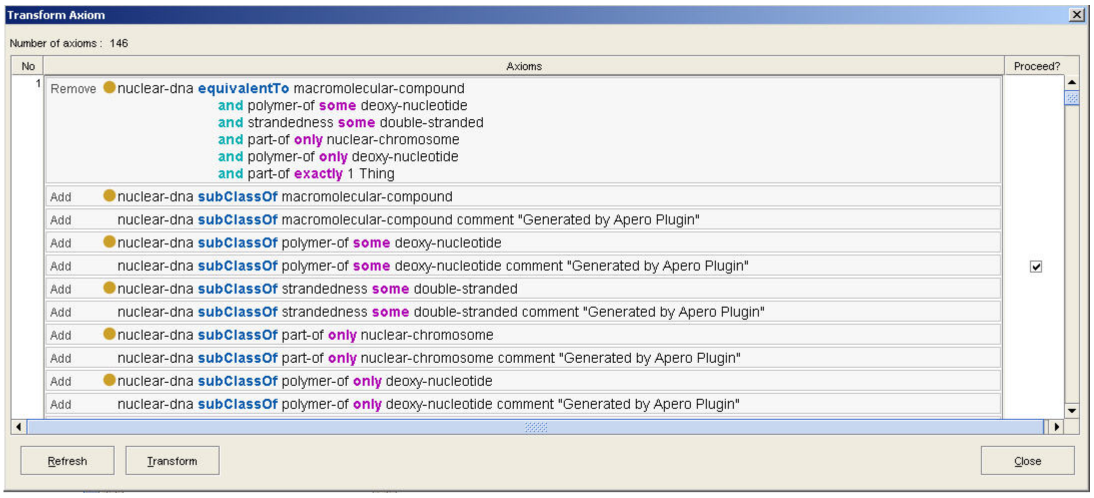
Table 7.2: Summary of testing result

Meanwhile, the Table 7.3 represents distribution of antipattern finding in all ontologies. Majority of known antipatterns is found in HydrOntology. This indicate that users of HydrOntology do not have enough knowledge about logic programming and how to write definition correctly.

No	Antipattern	Ontology			
		Computer Science	Tambis Full	Sweet Numeric	HydrOntology
DLAP					
1	AIO	0	0	0	2
2	OIL	0	0	0	0
3	OILWI	0	0	0	1
4	UE	0	0	0	0
5	UEWI	0	0	0	3
6	UEWPI	0	0	0	1
7	UEWIP	2	0	0	2
8	VOV	0	0	2	0
9	EID	0	0	0	6
10	EAD	1	3	0	0
11	SID	0	0	0	6
12	SAD	0	0	0	0
13	MMCAR	1	0	0	0
14	ECRWIP	0	0	0	3
15	SOSER	0	0	0	2
CLAP					
16	SOE	0	4	4	6
17	SOS	0	0	0	1
Guidelines					
18	DOC	0	0	0	0
19	UIE	0	0	0	1
20	ECR	0	4	0	3
21	SMALO	0	4	0	2
22	MIZ	0	0	0	1
23	DOS	0	27	1	3
24	DCS	0	0	3	3

Table 7.3: Detail of testing result

Another phenomena that appears on the Table 7.3 is the significant appearance occurred in *Tambis* ontology on DOS antipattern with 27 instances. The large number of instances indicates that users of the ontology need to be trained about how to write definitions. DOS antipattern may cause an antipattern in DLAP to be undetectable. Coincidently, it is also be supported by finding of 146 axioms on the transformation dialog (see the Figure 7.2) that has only one transformation rule almost similar to DOS. Especially for transformation, Users must confirm the correctness of each listed axiom with the real world, so that they are sure that transformation is needed.



Number of axioms : 146

No	Axioms	Proceed?
1	Remove ● nuclear-dna **equivalentTo** macromolecular-compound and polymer-of **some** deoxy-nucleotide and strandedness **some** double-stranded and part-of **only** nuclear-chromosome and polymer-of **only** deoxy-nucleotide and part-of **exactly** 1 Thing	
	Add ● nuclear-dna **subClassOf** macromolecular-compound	
	Add nuclear-dna **subClassOf** macromolecular-compound comment "Generated by Apero Plugin"	
	Add ● nuclear-dna **subClassOf** polymer-of **some** deoxy-nucleotide	
	Add nuclear-dna **subClassOf** polymer-of **some** deoxy-nucleotide comment "Generated by Apero Plugin"	✓
	Add ● nuclear-dna **subClassOf** strandedness **some** double-stranded	
	Add nuclear-dna **subClassOf** strandedness **some** double-stranded comment "Generated by Apero Plugin"	
	Add ● nuclear-dna **subClassOf** part-of **only** nuclear-chromosome	
	Add nuclear-dna **subClassOf** part-of **only** nuclear-chromosome comment "Generated by Apero Plugin"	
	Add ● nuclear-dna **subClassOf** polymer-of **only** deoxy-nucleotide	
	Add nuclear-dna **subClassOf** polymer-of **only** deoxy-nucleotide comment "Generated by Apero Plugin"	

Figure 7.2: Transformation Detection

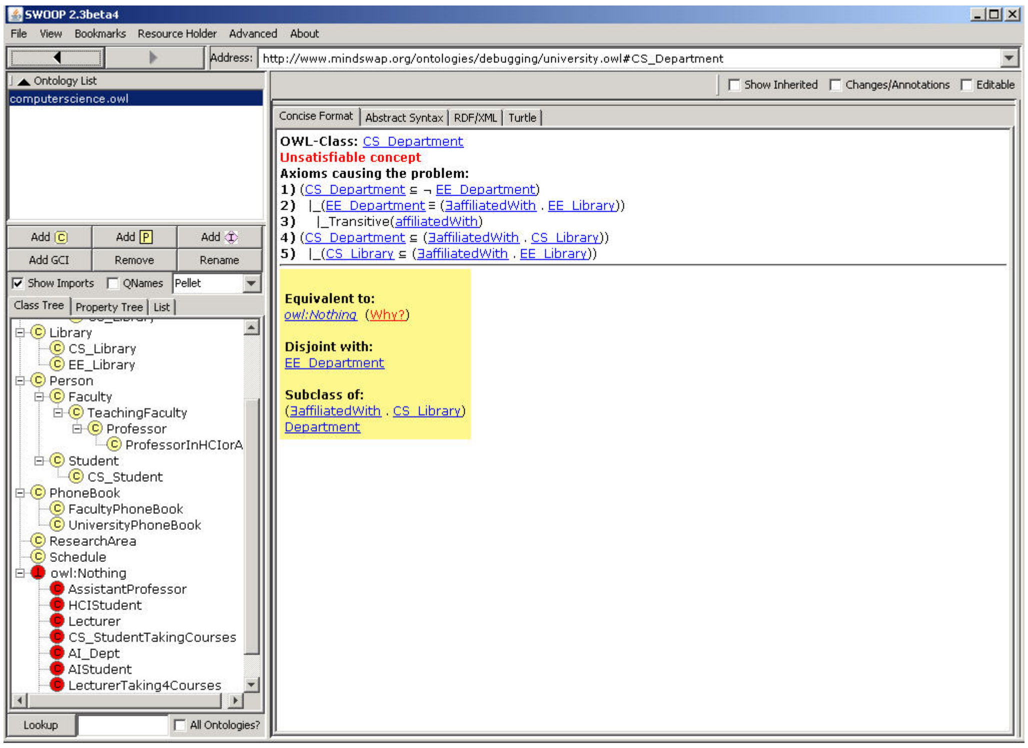
For equalization, we assume causes of unsatisfiability as instance of antipattern. After running all test cases on Apero and SWOOP, we found several things when debugging all ontologies as follows:

- Apero does not need a reasoner to detect an antipattern, but SWOOP needs it (Pellet reasoner) to determine whether a class satisfiable or not, and root or derived unsatisfiable class.
- SWOOP only detect causes of unsatisfiability (DLAP in Apero), but Apero also can detect another potential causes of error on CLAP and Guideline.
- Focusing on DLAP and manually we generate root of unsatisfiable class can be detected on Apero, the Table 7.4 show the detection result. Especially for HydrOntology, after SWOOP fails to debug, we do not continue comparison on root of unsatisfiable class.

Ontology	Response time (second)		Root of unsatisfiable class		
	Apero	SWOOP	Protege Reasoner	Apero	SWOOP
Computer_Science	<1	<1	5	4	5
Tambis_full	1	5	3	3	3
Sweet_Numeric	27	5	2	2	0
HydrOntology	3	Fail	Not continued		

Table 7.4: Comparison between Apero and SWOOP

The result on Computer_Science tells us to improve number of antipattern in Apero. Apero could not detect unsatisfiability of class *CS_Department* (see the Figure 7.3). This is also opportunity to define new antipattern with helped by another tool such as SWOOP. Meanwhile, detection of SWOOP


SWOOP 2.3beta4
File View Bookmarks Resource Holder Advanced About
Address:
Ontology List
Add C Add P Add I
Add GCI Remove Rename
Show Imports QNames Pellet
Class Tree Property Tree List
Show Inherited Changes/Annotations Editable
Concise Format Abstract Syntax RDF/XML Turtle
Lookup All Ontologies?

```
http://www.mindswap.org/ontologies/debugging/university.owl#CS_Department
```

computerscience.owl

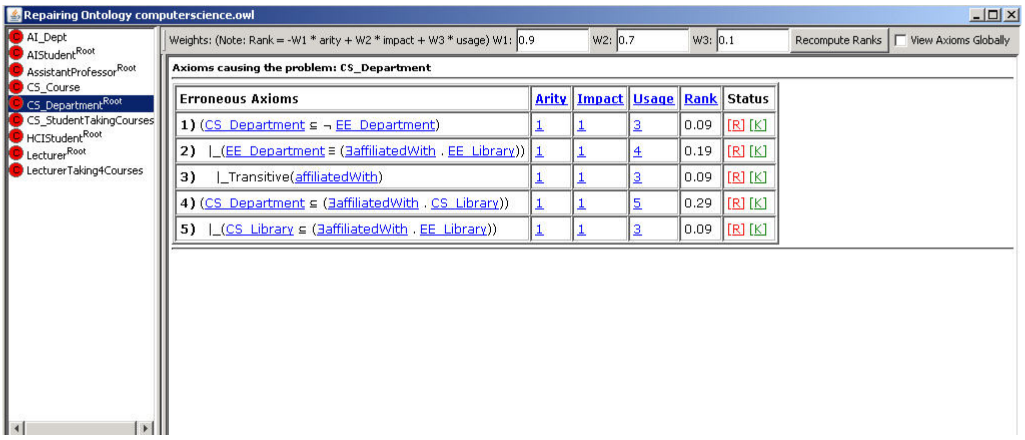
```
Library
    CS_Library
    EE_Library
Person
    Faculty
        TeachingFaculty
            Professor
                ProfessorInHCIorA
    Student
        CS_Student
PhoneBook
    FacultyPhoneBook
    UniversityPhoneBook
ResearchArea
Schedule
owl:Nothing
    AssistantProfessor
    HCIStudent
    Lecturer
    CS_StudentTakingCourses
    AI_Dept
    AIStudent
    LecturerTaking4Courses
```

```
OWL-Class: CS_Department
Unsatisfiable concept
Axioms causing the problem:
1) (CS_Department ⊆ ¬ EE_Department)
2)   |_(EE_Department ≡ (∃affiliatedWith . EE_Library))
3)     |_Transitive(affiliatedWith)
4) (CS_Department ⊆ (∃affiliatedWith . CS_Library))
5)   |_(CS_Library ⊆ (∃affiliatedWith . EE_Library))

Equivalent to:
owl:Nothing (Why?)

Disjoint with:
EE_Department

Subclass of:
(∃affiliatedWith . CS_Library)
Department
```

Figure 7.3: Detection of *CS_Department* on SWOOP

is better than Apero on Sweet_Numeric because SWOOP fails to load some indirect imported ontologies that are supposed to be imported by a direct imported one. SWOOP on this example is able to load 2 of 9 indirect imported ones. Therefore, SWOOP detects nothing because roots of unsatisfiable class probably occur in a failure imported ontology.

- SWOOP is powerful to give recommendation by ranking every involved axioms (see the Figure 7.4) but no explanation is given while Apero gives recommendation according experience of user.


Repairing Ontology computerscience.owl

AI_Dept
AIStudent^{Root}
AssistantProfessor^{Root}
CS_Course
CS_Department^{Root}
CS_StudentTakingCourses
HCIStudent^{Root}
Lecturer^{Root}
LecturerTaking4Courses

Weights: (Note: Rank = -W1 * arity + W2 * impact + W3 * usage) W1: 0.9 W2: 0.7 W3: 0.1 Recompute Ranks ☐ View Axioms Globally

Axioms causing the problem: CS_Department

Erroneous Axioms	Arity	Impact	Usage	Rank	Status
1) (CS_Department ⊆ ¬ EE_Department)	1	1	3	0.09	[R] [K]
2) |_(EE_Department ≡ (∃affiliatedWith . EE_Library))	1	1	4	0.19	[R] [K]
3) |_Transitive(affiliatedWith)	1	1	3	0.09	[R] [K]
4) (CS_Department ⊆ (∃affiliatedWith . CS_Library))	1	1	5	0.29	[R] [K]
5) |_(CS_Library ⊆ (∃affiliatedWith . EE_Library))	1	1	3	0.09	[R] [K]

Figure 7.4: Recommendation on SWOOP

Chapter 8

Conclusion and Future Work

Some work objectives have been accomplished by presentation on Chapter 4, 5 and 6. Moreover, we have done evaluation and confirmed some work objectives and all hypotheses fulfilled. We have achieved some bullet points of conclusion in this thesis as follows:

- This thesis has enriched the catalogue of antipattern from 10 antipatterns to 24 antipatterns. Some antipatterns have more than one detection pattern (AIO, UEWI, UEWPI, UEWIP and MMCaR antipattern). In total, we have collected 29 antipatterns.

- There are two additional symbols as representation of antipattern, namely $*$ (star) and $+$ (plus). We have seen these symbols as completed DL-symbols in formulating an antipattern especially to represent transitivity on subclass ($\sqsubseteq$) and equivalence ($\equiv$) relation.

- Both chapter 4 and 5 have classified implementation type of antipattern. The easy antipattern may be classified to OPPL-based antipattern and the rest must able to implemented by Lint, so we call it as Lint-based antipattern.

- Apero plug-in has been built to help the ontology developer to debug an ontology. Apero has implemented all antipatterns in this thesis and one transformation rule. Response time to debug an ontology on test cases is fast and there is no dependency to a reasoner.

- An antipattern has been a remedy to overcome difficulty in ontology debugging. In addition, a proposed debugging strategy is able to guide the ontology developer to solve inconsistency problem.

- By comparing between Apero and SWOOP, overall Apero proposes a better solution than SWOOP in ontology debugging for some reasons. Apero is more stable because Apero does not depend on a reasoner, unlike SWOOP does need a reasoner that somehow may not worked. Apero is able to detect not only unsatisfiability class but also modeling error and guideline. However, if reasoner works well, in some cases, SWOOP able to show number of unsatisfiability class better than Apero that depends on catalogue of antipattern.

However, from our limitation and conclusion, this thesis also leads us to some future works as follows:

- There are still a lot of antipattern that we need to identify. The catalogue of antipattern must still be improved. So far, we found an antipattern during debugging ontology manually. We also expect someone will discovery new antipatterns in better way.
- We expect in the next research that someone will consider naming convention of antipattern.
- Implementation of Lint-based antipattern is Java-based. There is always opportunity to improve an implementation such as inefficient code and more comprehensive testing to ensure validity of implementation.
- We suggest to debug a huge ontology to test the reliability of Apero. It is intended to find a bugs and idea for improvement.
- Discussion about debugging strategy based on antipatterns must be continued. It could be supported by performing evaluation by real user. It also opportunity to have automatic debugging strategy on Apero in a certain way.
- After introducing one transformation rule, for the next research, this topic may be exploit further to get new transformation rules. A next version of Apero is expected to have generic implementation of transformation.
- Implementation Lint-based antipattern exploits OWL API. It is a good idea to try implementation by exploiting SPARQL Query Language for RDF.

Bibliography

[1] *Reasoning for Ontology Engineering and Usage - ISWC 2008.* http://owl.cs.manchester.ac.uk/2008/iswc-tones/. [cited at p. 98]

[2] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, et al. OWL web ontology language reference. *W3C recommendation*, 10:2006–01, 2004. [cited at p. 49]

[3] Luis Manuel Vilches Blázquez, Miguel Ángel Bernabé Poveda, María del Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Antonio F. Rodríguez Pascual. Towntology & hydrontology: Relationship between urban and hydrographic features in the geographic information domain. In *Ontologies for Urban Development*, pages 73–84. 2007. [cited at p. 8]

[4] Oscar Corcho, Catherine Roussey, and Luis Manuel Vilches Blazquez. Catalogue of anti-patterns for formal ontology debugging. page 11, 2009. [cited at p. 7, 94, 95, 123]

[5] Oscar Corcho, Catherine Roussey, Luis Manuel Vilches Blazquez, and Ivan Perez. Pattern-based owl ontology debugging guidelines. page 11, 2009. [cited at p. 13, 14, 75, 124]

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, second edition edition, 2001. [cited at p. 32]

[7] Mikel Egaña, Robert Stevens, and Erick Antezana. Transforming the axiomisation of ontologies: The ontology pre-processor language. *In: Proceedings of OWLED 2008 DC OWL: Experiences and Directions*, page 10, 2008. [cited at p. 11]

[8] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in owl. In *Description Logics*, 2009. [cited at p. 5]

[9] Sattler U Horridge M, Parsia B. Laconic and pricise justifications in owl. *In: Proceedings of 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany*, LNCS 5318(323-338), 2008. [cited at p. 6, 8, 75]

[10] http://oppl2.sourceforge.net/grammar.html. Oppl grammar. [cited at p. ii, 111, 112, 114]

[11] http://protege.stanford.edu/. The protégé ontology editor and knowledge acquisition system. [cited at p. 10]

[12] http://www.cs.man.ac.uk/ iannonel/lintRoll/index.html. Lint detection for owl ontologies and related protégé plug-in. [cited at p. 14]

[13] http://www.w3.org/TR/owl guide/. Owl 2 web ontology language guide, 2004. [cited at p. 9, 11]

[14] Luigi Iannone, Mikel Egaña, Alan Rector, and Robert Stevens. Augmenting the expressivity of the ontology pre-processor language. *In: Proceedings of OWLED 2008 DC OWL:Experiences and Directions*, page 6, 2008. [cited at p. 11, 26]

[15] Sirin E Cuenca-Grau B. Kalyanpur A, Parsia B. Repairing unsatisfiable classes in owl ontologies. *In: Proceedings of 3rd European Semantic Web Conference (ESWC), Budva, Montenegero*, LNCS 4011(170-184), 2006. [cited at p. 6, 8, 75]

[16] Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. *Web Intelligence, IEEE / WIC / ACM International Conference on*, 0, 2006. [cited at p. 7]

[17] R. Raskin and M. Pan. Semantic web for earth and environmental terminology (sweet). In *Semantic Web Technologies for Searching and Retrieving Scientific Data*, 2003. [cited at p. 98]

[18] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. In *EKAW*, pages 63–81, 2004. [cited at p. 32, 71]

[19] R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N.W. Paton, C.A. Goble, and A. Brass. Tambis: Transparent access to multiple bioinformatics information sources. *Bioinformatics*, 16(2):184–186, 2000. [cited at p. 98]

[20] Heiner Stuckenschmidt. Debugging owl ontologies - a reality check. proceedings of the 6th international workshop on evaluation of ontology-based tools and the semantic web service challenge (eon-swsc-2008), tenerife, spain, june 1-2, 2008. In *EON*, volume 359 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. [cited at p. 8]

[21] Simon Thompson. *Haskell The Craft of Functional Programming*. Addison-Wesley, second edition edition, 1999. [cited at p. 33]

[22] L. M. Vilches Blázquez, M. A. B. Poveda, M. C. Suárez-Figueroa, A. Gómez-Pérez, and A. F. Rodríguez Pascual. Towntology & hydrontology: Relationship between urban and hydrographic features in the geographic information domain. In *Ontologies for Urban Development*, pages 73–84. 2007. [cited at p. 98]

[23] Horridge M Rector A Drummond N Seidenberg J. Wang, H. Debugging owl-dl ontologies: A heuristic approach. *In: Proceedings of 4th International Semantic Web Conference (ISWC), Galway, Ireland*, LNCS 3729(745-747), 2005. [cited at p. 5, 75]

Appendices

Appendix A

EBNF Production Rules for OPPL [10]

A.1 Statements

```
OPPL Statement ::= ( <VariableDeclaration> )? ( <Query> )? ( <Actions> )? ";"
VariableDeclaration ::= <VariableDefinition> ( "," <VariableDefinition> )*
Actions ::= "BEGIN" Action ( "," Action )+ "END"
VariableDefinition ::= <InputVariableDefinition> | <GeneratedVariableDefinition>
InputVariableDefinition ::= <IDENTIFIER> ":" <variableType> (<VariableTypeScope>)?
GeneratedVariableDefinition ::= <IDENTIFIER> ":" <variableType> "=" <opplFunction>
opplFunction ::= <create> | <creatInteserctions> | <createDisjunction>|
                 Any Manchester Syntax with variables expression compatible
                 with the generated variable.
create ::="create("<value>")"
createIntersection ::="createIntersection("<classvalues>")"
createDisjunction ::="createDisjunction("<classvalues>")"
/*The variable name in the production classvalues below must be of type CLASS*/
classvalues ::=<IDENTIFIER>".VALUES"
value ::= a string constant | <generatedValue>
generatedValue ::=<variableAttribute> (<aggregator> <variableAttribute>)*
aggregator ::="+"
variableAttribute ::=<IDENTIFIER>"."<attributeName>
attributeName ::="RENDERING"
VariableTypeScope ::= "[" <direction> <VariableFreeOWLExpression>"]"
direction ::= "subClassOf" | "superClassOf" | subPropertyOf | "superPropertyOf"
              | "instanceOf"
```

```
/*Direction production is not context free as it depends on which
variable type  the variable is being applied to. The scope, therefore,
is not context free either*/
Constraint ::= <IDENTIFIER> "!=" <OWLExpression>
              | <IDENTIFIER> "MATCH" <RegularExpression>
              | <IDENTIFIER> "IN" "{" <OWLExpression> ("," <OWLExpression>)* "}"
variableType ::= "CLASS" | "OBJECTPROPERTY | "DATAPROPERTY" | "INDIVIDUAL" | "CONSTANT
Query ::= "SELECT" ("ASSERTED")? <Axiom> ( ", ("ASSERTED")?" <Axiom> )*
          ( "WHERE" <Constraint> ("," <Constraint> )* )?
Action ::= "ADD" | "REMOVE" <Axiom>
Axiom ::= An axiom in Manchester OWL Syntax (possibly containing variables)
IDENTIFIER ::= "?"<LETTER> (<LETTER>|<DIGIT>)*
LETTER ::= ["_","a"-"z","A"-"Z", ,"\u00e0"-"\u00f9"]
DIGIT ::= ["0"-"9"]
OWLExpression ::= An OWL entity in Manchester OWL Syntax
                  (possibly containing variables)
VariableFreeOWLExpression ::= An OWL entity in Manchester OWL Syntax
                              (without variables)
RegularExpression ::= A Java regular expression for string matching
                      (applies to the entity rendering)
```

A.2 Manchester OWL Syntax axioms

```
SubClassAxiom ::= <ClassDescription> "SubClassOf" <ClassDescription>
EquivalentClassAxiom ::= <ClassDescription> "EquivalentTo" (<ClassDescription>)+
DisjointClassAxiom ::= <ClassDescription> "DisjointWith" (<ClassDescription>)+
FunctionalObjectPropertyAxiom ::= "Functional" <ObjectProperty>
SymmetricObjectPropertyAxiom ::= "Symmetric" <ObjectProperty>
ReflexiveObjectPropertyAxiom ::= "Reflexive" <ObjectProperty>
TransitiveObjectPropertyAxiom ::= "Transitive" <ObjectProperty>
AntiSymmetricObjectPropertyAxiom ::= "AntiSymmetric" <ObjectProperty>
IrreflexiveObjectPropertyAxiom ::= "Irreflexive" <ObjectProperty>
SubObjectPropertyAxiom ::= <ObjectProperty> "SubPropertyOf" <ObjectProperty>
EquivalentObjectPropertyAxiom ::= <ObjectProperty> "EquivalentTo" (<ObjectProperty>)+
DisjointPropertyAxiom ::= <ObjectProperty> "DisjointWith" (<ObjectProperty>)+
InversePropertyAxiom ::= <ObjectProperty> "InverseOf" "("<ObjectProperty>")"
InverseFunctionalPropertyAxiom ::= <ObjectProperty> "InverseFunctional"
                                   "("<ObjectProperty>")"
FunctionalDataPropertyAxiom ::= "Functional" <DataProperty>
ObjectPropertyRangeAxiom ::= <ObjectProperty> "Range" <ClassDescription>
```

```
ObjectPropertyDomainAxiom ::= <ObjectProperty> "Domain" <ClassDescription>
SubDataPropertyAxiom ::= <DataProperty> "SubPropertyOf" <DataProperty>
EquivalentDataPropertyAxiom ::= <DataProperty> "EquivalentTo" (<DataProperty>)+
DisjointPropertyAxiom ::= <DataProperty> "DisjointWith" (<DataProperty>)+
DataPropertyDomainAxiom ::= <DataProperty> "Domain" <ClassDescription>
DataPropertyRangeAxiom ::= <DataProperty> "Range" <DataRange>
ClassAssertionAxiom ::= <ClassDescription> <Individual>
ObjectPropertyAssertionAxiom ::= <Individual> <ObjectProperty> <Individual>
DataPropertyAssertionAxiom ::= <Individual> <DataProperty> <Constant>
NegativeObjectPropertyAssertionAxiom ::= "not" <Individual> <ObjectProperty>
                                         <Individual>
NegativeDataPropertyAssertionAxiom ::= "not" <Individual> <DataProperty> <Constant>
SameAsAxiom ::= <Individual> "sameAs" (<Individual>)+
DifferentFromAxiom ::= <Individual> "differentFrom" (<Individual>)+
```

A.3 Manchester OWL Syntax with variables entities

```
 ClassDescription ::= <ClassIntersection>
ClassIntersection ::= <ClassUnion> ("and" <ClassUnion>)*
ClassUnion ::= <NonN-aryDescription> ("or " <NonN-aryDescription>)*
NonN-aryDescription ::= <PrimitiveClass> | <ObjectRestriction>
                        | <DataRestriction> | "not" <ClassDescription>
                        | "oneOf {" <Individual> (, <Individual>)* "}"
DataRestriction ::= <DataProperty> "some" <DataRange> | <DataProperty> "only" <DataRa
ObjectRestriction ::= <ObjectProperty> "some" <ClassDescription>
                    | <ObjectProperty> "only" <ClassDescription>
                    | <ObjectProperty> "value" <Individual>
                    | <ObjectProperty> "min" <NonNegativeInteger> (<ClassDescriptio
                    | <ObjectProperty> "exactly" <NonNegativeInteger> (<ClassDescri
                    | <ObjectProperty> "max" <NonNegativeInteger> (<ClassDescriptio
PrimitiveClass ::=<ClassName> | <VariableName>
ObjectProperty ::=<ObjectPropertyName> | <VariableName>
DataProperty ::=<DataPropertyName> | <VariableName>
Individual ::=<IndividualName> | <VariableName>
Constant ::=<ConstantLiteral> | <VariableName>
ClassName ::= <LETTER> (<LETTER>|<DIGIT>)*
ObjectPropertyName ::= <LETTER> (<LETTER>|<DIGIT>)*
DataPropertyName ::= <LETTER> (<LETTER>|<DIGIT>)*
IndividualName ::= <LETTER> (<LETTER>|<DIGIT>)*
ConstantLiteral ::= "See the OWL specification"
```

```
DataRange ::= See Manchester OWL Syntax references above
NonNegativeInteger ::= Any integer greater than or equal to zero
ClassName ::= <LETTER> (<LETTER>|<DIGIT>)*
VariableName ::= "?" <LETTER> (<LETTER>|<DIGIT>)*
LETTER ::= ["_","a"-"z","A"-"Z","\u00e0"-"\u00f9"]
DIGIT ::= ["0"-"9"]
```

Appendix B

List of Method in OWLFunc Class

1.
   ```
   boolean isSuccessorOf(OWLOntology ontology,
          Set<OWLOntology> importedOntologies, OWLClass c1, OWLClass c2)
   ```

   This method checks whether $c_1 \sqsubseteq c_2$ in ontologies. [*]

2.
   ```
   boolean generateAncestorPath(LinkedList<OWLAxiom> curPath,
          OWLDataFactory dataFactory, OWLOntology ontology,
          Set<OWLOntology>  importedOntologies, OWLClass c1, OWLClass c2)
   ```

   This method generates path (list of axiom) that performs $c_1 \sqsubseteq^+ c_2$ in ontologies. The path will be returned in `curPath`. If there is no path, it returns false.

3.
   ```
   boolean generateAncestorPathToAnonymous(
         LinkedList<OWLAxiom> curPath, OWLDataFactory dataFactory,
         OWLOntology ontology, Set<OWLOntology> importedOntologies,
         OWLClass c1, OWLDescription c2)
   ```

   This method is almost similar to the previous method, but c_2 must be anonymous class.

4.
   ```
   Set<OWLClass> getAllDescendent(OWLOntology ontology,
         Set<OWLOntology> importedOntologies, OWLClass c1)
   ```

   This method returns all descendent class c in $c \sqsubseteq^+ c_1$.

5.
   ```
   Set<OWLClass> getIntersectionClass(Set<OWLClass> g1, Set<OWLClass> g2)
   ```

   This method returns intersection between two set of OWLClass.

[*]Ontologies means $\{\text{ontology}\} \sqcup$ importedOntologies

6. `Set<OWLDescription> getIntersectionDescription(`
   `Set<OWLDescription> g1, Set<OWLDescription> g2)`

   This method returns intersection between two set of OWLDescription.

7. `Set<OWLEquivalentClassesAxiom> getOWLEquivalentClassesAxioms(`
   `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

   This method returns all equivalent class axioms in ontologies.

8. `Map<OWLEquivalentClassesAxiom,OWLOntology>`
   `getOWLEquivalentClassesAxiomsAndOntology(`
   `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

   This method returns all equivalent class axioms in ontologies including a mapping of each axiom to its ontology.

9. `Set<OWLSubClassAxiom> getOWLSubClassAxioms(OWLOntology curOntology,`
   `Set<OWLOntology> importedOntologies)`

   This method returns all subclass axioms in ontologies.

10. `Set<OWLSubClassAxiom> getSubClassAxiomsForLHS(OWLClass cls,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

    This method returns all subclass axioms in ontologies whose the subclass is cls.

11. `Set<OWLObjectSubPropertyAxiom> getObjectSubPropertyAxiomsForLHS(`
    `OWLObjectPropertyExpression r,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

    This method returns all sub object property axioms in ontologies whose the sub property is r.

12. `Set<OWLObjectSubPropertyAxiom> getObjectSubPropertyAxiomsForRHS(`
    `OWLObjectPropertyExpression r,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

    This method returns all sub object property axioms in ontologies whose the super property is r.

13. `Set<OWLObjectPropertyExpression> getObjectSubProperty(`
    `OWLObjectPropertyExpression r,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

    This method returns all sub object properties of r in ontologies.

14. `Set<OWLObjectPropertyExpression> getObjectPropertyInverse(`
    `OWLObjectPropertyExpression r,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

This method returns all inverse object properties of r in ontologies.

15. `Set<OWLObjectPropertyExpression> getObjectSuperProperty(`
    ```
    OWLObjectPropertyExpression r,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all super object properties of r in ontologies.

16. `Set<OWLObjectPropertyExpression> getObjectSubPropertyClosure(`
    ```
    OWLOntology ontology,
    Set<OWLOntology> importedOntologies, OWLObjectPropertyExpression r)
    ```
    This method returns all sub object properties of r in ontologies, taking into account an indirect sub property.

17. `Set<OWLDisjointClassesAxiom> getDisjointClassesAxioms(OWLClass cls,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all disjoint class axioms that contain cls.

18. `Set<OWLDescription> getDisjointClasses(OWLClass cls,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all classes (including anonymous class) that are disjoint with cls.

19. `Set<OWLEquivalentClassesAxiom> getEquivalentClassesAxioms(OWLClass cls,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all equivalent class axioms that contain cls.

20. `Set<OWLEquivalentObjectPropertiesAxiom> getEquivalentPropertyAxioms(`
    ```
    OWLObjectPropertyExpression r,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all equivalent object property axioms that contain r.

21. `Set<OWLEquivalentClassesAxiom> getEquivalentClassesAxioms(OWLClass cls1,`
    ```
    OWLClass cls2,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies,
    Set<OWLEquivalentClassesAxiom> axiomChecked)
    ```
    This method returns all equivalent class axioms that perform $cls_1 \equiv^+ cls_2$.

22. `Set<OWLDescription> getEquivalentClassClosure(OWLClass cls1,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies,
    Set<OWLEquivalentClassesAxiom> axiomChecked)
    ```
    This method returns all classes that are equivalent with cls_1 directly or indirectly (by transitivity of $\equiv$).

23. `Set<OWLObjectPropertyExpression> getEquivalentPropertyClosure(`
    ```
    OWLObjectPropertyExpression r,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies,
    Set<OWLEquivalentObjectPropertiesAxiom> axiomChecked)
    ```
    This method returns all object properties that are equivalent with r directly or indirectly (by transitivity of $\equiv$).

24. `Set<OWLDescription> getEquivalentClassClosureWithAnonymous(`
    ```
    OWLDescription cls1,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies,
    Set<OWLEquivalentClassesAxiom> axiomChecked)
    ```
    This method returns all classes (including anonymous class)that are equivalent with cls_1 directly or indirectly (by transitivity of $\equiv$).

25. `boolean generateEquivalentPath(`
    ```
    LinkedList<OWLEquivalentClassesAxiom> curPath,
    OWLClass c1, OWLDescription c2,
    OWLOntology curOntology, Set<OWLOntology> importedOntologies,
    Set<OWLEquivalentClassesAxiom> axiomChecked)
    ```
    This method returns all equivalent class axiom that perform $c_1 \equiv^+ c_2$ where c_2 may be an anonymous class.

26. `Set<OWLDescription> getSuperClasses(OWLClass cls,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all super classes of cls in ontologies.

27. `Set<OWLDescription> getEquivalentClasses(OWLClass cls,`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all classes that are equivalent with cls in ontologies.

28. `Set<OWLFunctionalDataPropertyAxiom> getOWLFunctionalDataPropertyAxiom(`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all functional data property axiom in ontologies.

29. `Set<OWLFunctionalObjectPropertyAxiom> getOWLFunctionalObjectPropertyAxiom(`
    ```
    OWLOntology curOntology, Set<OWLOntology> importedOntologies)
    ```
    This method returns all functional object property axiom in ontologies.

30. `boolean contentAxiom(OWLOntology curOntology,`
    ```
    Set<OWLOntology> importedOntologies, OWLAxiom axiom)
    ```
    This method checks whether ontologies contain *axiom*.

31. `LinkedList<OWLAxiom> getDisjointLink(OWLDataFactory dataFactory,`
```
      OWLDescription d1, OWLDescription d2,
      OWLOntology curOntology, Set<OWLOntology> importedOntologies)
```
This method return all axioms that prove disjointness between d_1 and d_2.

32. `LinkedList<OWLAxiom> getDisjointLinkOnUnion(`
```
       OWLDataFactory dataFactory, Set<OWLClass> s1, Set<OWLClass> s2,
       OWLOntology curOntology, Set<WLOntology> importedOntologies)
```
This method return all axioms that prove disjointness between d_1 and d_2 where d_i is set of operand in union operator.

33. `LinkedList<OWLAxiom> getDisjointLinkAmongIntersection(`
```
      OWLDataFactory dataFactory, Set<OWLDescription> classOperands,
      OWLOntology curOntology, Set<OWLOntology> importedOntologies)
```
This method return all axioms proving that there is one disjointness among classes in *classOperands*.

34. `LinkedList<OWLAxiom> getDisjointLink(`
```
      OWLDataFactory dataFactory, OWLClass c1, OWLClass c2,
      OWLOntology curOntology, Set<OWLOntology> importedOntologies)
```
This method return all axioms that show disjointness between class c_1 and c_2.

35. `Set<OWLClass> getAllAncestor(OWLOntology ontology,`
```
      Set<OWLOntology> importedOntologies, OWLClass c1)
```
This method returns all classes where each class c in $c_1 \sqsubseteq^+ c$.

36. `Set<OWLDescription> getAllAnonymousAncestor(`
```
      OWLOntology ontology, Set<OWLOntology> importedOntologies, OWLClass c1)
```
This method returns all anonymous classes where each class c in $c_1 \sqsubseteq^+ c$.

37. `LinkedList<OWLAxiom> getFirstAxiomsHavingMinOrExactOrSomeRestriction(`
```
      OWLDataFactory dataFactory,
      OWLOntology ontology, Set<OWLOntology> importedOntologies,
      OWLClass c1, OWLObjectPropertyExpression r)
```
This method return all axioms that show property r must have at least a value , normally specified as existential restrictions, (minimum) or exact cardinality restriction for class c_1 with a positive number on the cardinality.
.

38. `boolean isConceptExist(OWLClass c1,OWLOntology curOntology)`

This method check whether class c_1 exists in the *curOntology*.

39. `int countFreqConcept(OWLClass c1,`
    `OWLOntology curOntology, Set<OWLOntology> importedOntologies)`

    This method return frequency of class c_1 used in ontologies.

40. `Set<OWLAnnotation> getClassAnnotation(OWLClass cls,`
    `OWLOntology curOntology)`

    This method return all annotations of class cls.

List of Symbols and Abbreviations

Abbreviation	Description	Definition
OWL	Ontology Web Language	page 5
DL	Description Logic	page 5
OPPL	Ontology Pre-Processor Language	page 11
RDF	Resource Description Framework	page 10
API	Application Programming Interface	page 11
Jar	Java Archive	page 83
GUI	Graphical User Interface	page 88

List of Figures

List of Tables

List of Codes