Towards a Set of OWL Ontology Debugging Guidelines

Catherine Roussey
Université de Lyon CNRS
Université Lyon 1, LIRIS
UMR5205
Villeurbanne, France
catherine.roussey@liris.cnrs.fr

Oscar Corcho
Ontology Engineering Group.
Departamento de Inteligencia
Artificial. Universidad Politécnica
de Madrid Spain
ocorcho@fi.upm.es

Luis Manuel Vilches Blázquez
Ontology Engineering Group.
Departamento de Inteligencia
Artificial. Universidad Politécnica
de Madrid, Spain
lmvilches@ fi.upm.es

ABSTRACT
Debugging inconsistent OWL ontologies is normally a tedious and time-consuming task where a combination of ontology engineers and domain experts is often required to understand whether the changes performed are actually dealing with formalisation errors or changing the intended meaning of the original knowledge model. Debugging services included in existing ontology engineering tools aid in this task. However, in complex cases they are still far from providing adequate support to ontology developers, due to their lack of efficiency or precision when explaining the main causes for inconsistencies. We claim that it is possible to provide additional guidelines to these users, based on the identification of common antipatterns and an adequate debugging strategy, which can be combined with the use of these tools in order to make this task more effective.

Categories and Subject Descriptors
I.2.4 Knowledge Representation Formalisms and Methods – *representation languages*

General Terms
Design, Languages, Verification

Keywords
OWL, ontology, debugging, antipattern

1. INTRODUCTION
Ontology engineering methodologies describe the sets of activities to be carried out for ontology building, together with methods and techniques that can be applied to them. Among these activities, those of ontology formalisation and implementation appear in most methodologies, since the final objective is to obtain one or several ontologies that describe the domain according to the ontology requirement specifications provided in the early stages of development.

Formalisation and implementation activities have different degrees of difficulty, considering the knowledge representation formalism and ontology language selected, and the ontology requirements, among others. It is not the same to formalise and implement an RDF(S) ontology than an OWL ontology. Nor is it the same to develop a small OWL ontology where only primitive concepts are needed than a network of ontologies where defined concepts are extensively used and complex inferences have to be drawn upon.

These degrees of difficulty can be somehow transitively applied to most of the tasks in which these activities can be decomposed, one of which is ontology debugging. There are no studies about how difficult it is nor about how much effort it takes to debug an ontology (not even in approaches like ONTOCOM [8]). Taking into account the experience from debugging in software engineering and in knowledge representation systems, we can intuitively assume that this is one of the most effort-consuming tasks. Therefore any help for ontology developers for this task will be probably translated into a reduction in the effort and time required to obtain results, and a better quality of the results.

If we focus on description logic formalisation and OWL implementation, several debugging tools exist ([4,5]). These tools isolate as much as possible the axioms “responsible” for inconsistencies, and find the roots of inconsistency problems, which are then propagated throughout the concept hierarchy. These tools have proven their effectiveness, but are still far from optimal in the provision of adequate explanations about the reasons for inconsistencies and in the proposal of alternatives to resolve the identified conflicts. To illustrate this, we experienced with early versions of a medium-sized OWL ontology developed by a domain expert in the area of hydrology [15], consisting of 150 classes, 34 object properties and 66 datatype properties[1]. The first version generated by a group of experts had a total of 102 unsatisfiable classes. The debugging systems used ([4,5]) did not provide enough information about root unsatisfiable classes or adequate (e.g., understandable by domain experts) justifications of the reasons for their unsatisfiability. And in several occasions during the debugging process the generation of justifications for inconsistencies took several hours, what made these tools hard to use. As a result, we found out that in several occasions domain experts were just changing axioms from the original ontology in a somehow random manner, even changing the intended meaning of the real definitions instead of correcting errors in their formalisations.

We made an effort to understand common inconsistency-leading patterns used by domain experts when implement-

[1] This early version of the ontology is available at http://www.dia.fi.upm.es/~ocorcho/OWLDebugging/, together with some of the most relevant intermediate versions of such ontology as domain experts were debugging it.

ing OWL ontologies, based on existing ontology design patterns and knowledge patterns and anti-patterns.

In this paper we propose a detailed list of such anti-patterns, compiling all the relevant cases that we came across when helping ontology developers to debug their ontologies. Then we provide some hints about how to organise the iterative ontology debugging process using a combination of debugging tools and patterns, and we describe the most common alternatives to tackle these anti-patterns.

2. PATTERNS AND ANTI-PATTERNS

In software engineering, a design pattern can be defined as a general, proven and beneficial solution to a common reoccurring problem in software design [2]. Built upon similar experiences, design patterns represent best practices about how to build software. On the contrary, antipatterns are defined as patterns that appear obvious but are ineffective or far from optimal in practice, representing worst practice about how to structure and build software [6].

In knowledge (and more specifically in ontology) engineering the concept of knowledge modelling (ontology design) pattern is used to refer to modelling solutions that allow solving recurrent knowledge modelling or ontology design problems [1, 9, 10, 11]. A similar definition is given for knowledge modelling (or ontology design) anti-patterns.

Different types of ontology design patterns are defined [9]:

Logical Ontology Design Patterns (LP) are independent from a specific domain of interest, but dependent on the expressivity of the logical formalism used for representation. For example, the n-ary relation pattern enables to model n-ary relations in OWL DL ontologies ([9, 17]).

Architectural Ontology Design Patterns (AP) provide recommendations about the structure of an ontology. They are defined in terms of LPs or compositions of them. Examples are: taxonomy or lightweight ontology [9].

Content Ontology Design Patterns (CP) propose domain-dependent conceptual models to solve content design problems for the domain classes and properties that populate an ontology. They usually exemplify LPs, and they represent most of the work done on ontology design patterns.

In contrast to ontology design patterns, the work on anti-patterns is less detailed ([7, 12]). Four LAPs are presented in [7], all of them focused on property domains and ranges. And [12] describes common difficulties for newcomers to description logics in understanding the logical meaning of expressions. However, none of these contributions groups anti-patterns in a common classification, nor provide a comprehensive set of hints to debug them.

2.1 A Classification of Ontology Design Anti-Patterns

We have identified a set of patterns that are commonly used by domain experts in their DL formalisations and OWL implementations, and that normally result in inconsistencies. We have categorized them into three groups:

- Logical Anti-Patterns (LAP). They represent errors that DL reasoners detect. These are the ones for which tool support is easier to provide and hence some support already exists.
- Non-Logical (aka Cognitive) Anti-patterns (NLAP). They represent possible modelling errors that are not detected by reasoners (they are not logical but modelling errors, which 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 a logical point of view, but in which the ontology developer could have used other simpler alternatives for encoding the same knowledge.

In the rest of this section we describe each of the anti-patterns identified in each group, providing their name and acronym, their template logical expressions and a brief explanation of why this anti-pattern can appear and how it should be checked by the ontology developer. As aforementioned, it is important to note that LAP are identified by existing ontology debugging tools, although the information that is provided back to the user explaining the reason for the inconsistency is not described according to such a pattern, what makes it difficult for ontology developers to find out where the inconsistencies are coming from. With respect to NLAP and G, they are not currently detected by these tools as such, although in some cases their combination may lead into inconsistencies that are detected (although not appropriately explained) by tools. As we mention in our future work section, we think that tool support for them could be a major step forward in this task.

Finally, all these anti-patterns should be seen as elementary units that cause ontology inconsistencies. That is, they can be combined into more complex ones. However, providing a solution for the individual ones is already a good advance to the current state of the art, and our future work will be also devoted to finding the most common combinations and providing recommendations for them.

2.2 Logical Antipatterns

AntiPattern AndIsOr (AIO)

$C1 \sqsubseteq \exists R.(C2 \sqcap C3)$, disj(C2,C3)[2]

This is a common modelling 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 [12]. For example, I want a cake with milk and chocolate is ambiguous. Does the cake recipe contain?

[2] This does not mean that the ontology developer has explicitly expressed that C2 and C3 are disjoint, but that these two concepts are determined as disjoint from each other by a reasoner. We use this notation as a shorthand for $C2 \sqcap C3 \sqsubseteq \bot$.

- ...Some chocolate plus some milk?
  CakeRecipe $\subseteq$ *∃contain.Chocolate and ∃contain.Milk*
- ...Chocolate-flavoured milk?
  CakeRecipe $\subseteq$ *∃contain.(Chocolate*$\cap$*Milk)*
- ...Some chocolate or some milk?
  CakeRecipe $\subseteq$ *∃contain.(Chocolate*$\cup$*Milk)*

AntiPattern OnlynessIsLoneliness (OIL)

$C1 \subseteq \forall R.C2$, $C1 \subseteq \forall R.C3$, disj(C2,C3)

The ontology has a universal restriction to say that C1 can only be linked with property R to C2. Next, a new universal restriction is added saying that C1 can only be linked with property R to C3, disjoint with C2. In general, this means that the ontology developer forgot the previous axiom.

AntiPattern OnlynessIsLonelinessWithInheritance (OILWI)

$C1 \subseteq C2$, $C1 \subseteq \forall R.C3$, $C2 \subseteq \forall R.C4$, disj(C3,C4)

The ontology developer has added a universal restriction for class C1 without remembering that he had already defined another universal restriction with the same property in a parent class. This anti-pattern is a specialization of OIL.

AntiPattern OnlynessIsLonelinessWithPropertyInheritance (OILWPI)

$R1 \subseteq R2$, $C1 \subseteq \forall R1.C2$, $C1 \subseteq \forall R2.C3$, disj(C2,C3)

The ontology developer misunderstands the subproperty relation between roles, thinking that it is similar to a part-of relation. This anti-pattern is a specialization of OIL because $C1 \subseteq \forall R1.C2, R1 \subseteq R2 \models C1 \subseteq \forall R2.C2$.

AntiPattern UniversalExistence (UE)

$C1 \subseteq \forall R.C2$, $C1 \subseteq \exists R.C3$, disj(C2,C3)

The ontology developer has added an existential restriction for a concept without remembering the existence of an inconsistency-leading universal restriction for that concept.

AntiPattern UniversalExistenceWithInheritance (UEWI)

$C1 \subseteq C2$, $C1 \subseteq \exists R.C3$, $C2 \subseteq \forall R.C4$, disj(C3,C4) or
$C1 \subseteq C2$, $C1 \subseteq \forall R.C3$, $C2 \subseteq \exists R.C4$, disj(C3,C4)

The ontology developer has added an existential/universal restriction in a concept without remembering that there was already an inconsistency-leading universal/existential restriction in a parent class, respectively. This anti-pattern is a specialization of UE.

AntiPattern UniversalExistenceWithPropertyInheritance1 (UEWPI)

$R1 \subseteq R2$, $C1 \subseteq \exists R1.C2$, $C1 \subseteq \forall R2.C3$, disj(C2,C3)

The ontology developer misunderstands the subproperty relation between roles, thinking that it is similar to a part-of relation. This anti-pattern is a specialization of UE because $C1 \subseteq \exists R1.C2, R1 \subseteq R2 \models C1 \subseteq \exists R2.C2$.

AntiPattern UniversalExistenceWithInverseProperty (UEWIP)

$C2 \subseteq \exists R^{-1}.C1$, $C1 \subseteq \forall R.C3$, disj(C2, C3)

The ontology developer has added restrictions about C2 and C1 using a role and its inverse. This antipattern is a specialization of UE because: $C2 \subseteq \exists R^{-1}.C1 \models C1 \subseteq \exists R.C2$.

AntiPattern EquivalenceIsDifference (EID)

$C1 \equiv C2$, disj(C1,C2)

This inconsistency comes from the fact that the ontology developer wants to say that C1 is a subclass of C2 (that is, that C1 is a C2, but at the same time it is different from C2 since he has more information). This anti-pattern is only common for ontology developers with no previous training in OWL modelling, since after a short training session they would discover that they really want to express $C1 \subseteq C2$.

2.3 Non Logical Anti-Patterns

As aforementioned, these anti-patterns are not necessarily errors, but describe common templates that ontology developers use erroneously trying to represent a different piece of knowledge.

AntiPattern SynonymeOfEquivalence (SOE)

$C1 \equiv C2$

The ontology developer wants to express that two classes C1 and C2 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 C1 has two labels: C1 and C2. Usually one of the classes is not used anywhere else in the axioms defined in the ontology.

AntiPattern SumOfSome (SOS)

$C1 \subseteq \exists R.C2$, $C1 \subseteq \exists R.C3$, disj(C2,C3)

The ontology developer has added a new existential restriction without remembering that he has already defined another existential restriction for the same concept and role. Although this could be ok in some cases (e.g., a child has at least one mother and at least one father), in many cases it represents a modelling error.

AntiPattern SumOfSomeWithInheritance (SOSWI)

$C1 \subseteq C2$, $C1 \subseteq \exists R.C3$, $C2 \subseteq \exists R.C4$, disj(C3,C4)

This anti-pattern is a specialization of SOS.

AntiPattern SumOfSomeWithPropertyInheritance (SOSWPI)

$R1 \subseteq R2$, $C1 \subseteq \exists R1.C2$, $C1 \subseteq \exists R2.C3$, disj(C2,C3)

The ontology developer misunderstands the subproperty relation between roles, thinking that it is similar to a part-of relation. This anti-pattern is a specialization of SOS because $C1 \subseteq \exists R1.C2$, $R1 \subseteq R2 \models C1 \subseteq \exists R2.C2$.

AntiPattern SumOfSomWithInverseProperty (SOSWIP)

$C2 \subseteq \exists R^{-1}.C1$, $C1 \subseteq \exists R.C3$, $disj(C2,C3)$

The ontology developer has created two existential restrictions using a role and its inverse. This anti-pattern specializes SOS because: $C2 \subseteq \exists R^{-1}.C1 \models C1 \subseteq \exists R.C2$.

AntiPattern SomeMeansAtLeastOne (SMALO)

$C1 \subseteq \exists R.C2$, $C1 \subseteq (\geq 1\ R.T)$

The cardinality restriction is superfluous.

2.4 Guidelines

As aforementioned, guidelines represent complex expressions used in an ontology component definition that are correct from a logical point of view, but in which the ontology developer could have used other simpler alternatives for encoding the same knowledge.

Guideline DisjointnessOfComplement (DOC)

$C1 \equiv \text{not } C2$

The ontology developer wants to say that C1 and C2 cannot share instances. Even if the axiom is correct from a logical point of view, it is more appropriate to state that C1 and C2 are disjoint.

Guideline Domain&CardinalityConstraints (DCC)

$C1 \subseteq \exists R.C2$, $C1 \subseteq (=2R.T)$

Ontology developers with little background in formal logic find difficult to understand that “only” does not imply “some” [12]. This antipattern is a counterpart of that fact. Developers may forget that existential restrictions contain a cardinality constraint: $C1 \subseteq \exists R.C2 \models C1 \subseteq (\geq 1R.C2)$. Thus, when they combine existential and cardinality restrictions, they may be actually thinking about universal restrictions with those cardinality constraints.

Guideline GroupAxioms (GA)

$C1 \subseteq \forall R.C2$, $C1 \subseteq (\geq 2R.T)$ (just as an example)

In order to facilitate the comprehension of complex class definitions, we recommend grouping all the restrictions of a concept that use the same role R in a single restriction.

Guideline MinIsZero (MIZ)

$C1 \subseteq (\geq 0R.T)$

The ontology developer wants to say that C1 can be the domain of the R role. This restriction has no impact on the logical model being defined and can be removed.

3. ONTOLOGY DEBUGGING STRATEGY

As mentioned in the introduction, OWL ontology debugging features have been proposed in the literature with different degrees of formality ([4, 5, 16]). These features are useful to debug ontologies and allow identifying the main roots for inconsistencies and the superfluous axioms and restrictions, and explaining them in some cases, with different degrees of detail, so that the debugging process can be guided by them and can be made more efficient. However, in general these features are mainly focused on the explanations of the logical entailments and are not so focused on the ontology engineering side of ontology debugging. Hence explanations are still difficult to understand for ontology developers. Furthermore, there are no clear strategies about how an ontology developer should debug inconsistent ontologies, in terms of steps to be followed in this process. Consequently, we think that there is a need to complement both types of suggestions in order to make the ontology debugging process more efficient.

In the following sections we describe the lifecycle that we propose in order to perform this task and the specific recommendations that we provide users for each of the anti-patterns defined in the previous section.

3.1 A Lifecycle for Ontology Debugging

Figure 1 shows graphically a usual ontology debugging lifecycle, with the roles of knowledge engineer and domain expert identified, although in many cases they are performed by the same person. As it happens in other disciplines (e.g., software development), the first step is to locate where the problems are, so as to propose recommendations for corrections, such as the ones presented in the section 3.2. Recommendations cannot always be automated, since they may change the intended meaning of the ontology, and should be documented. Once a change is done, new consistency checks should be performed. This is an iterative process to be followed until there are no more inconsistencies found in the ontology.

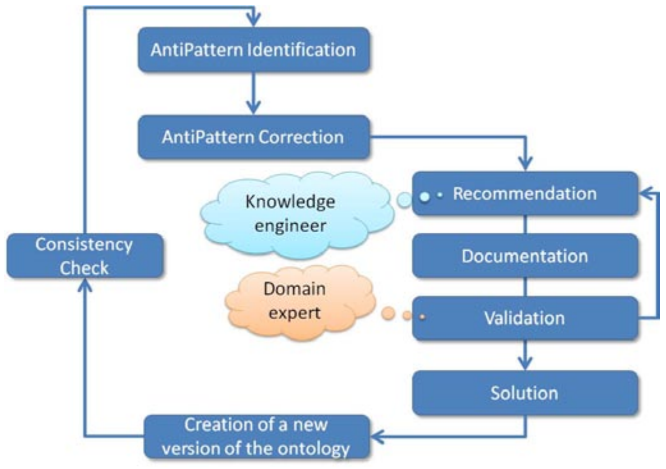

Figure 1. Global strategy for ontology debugging.

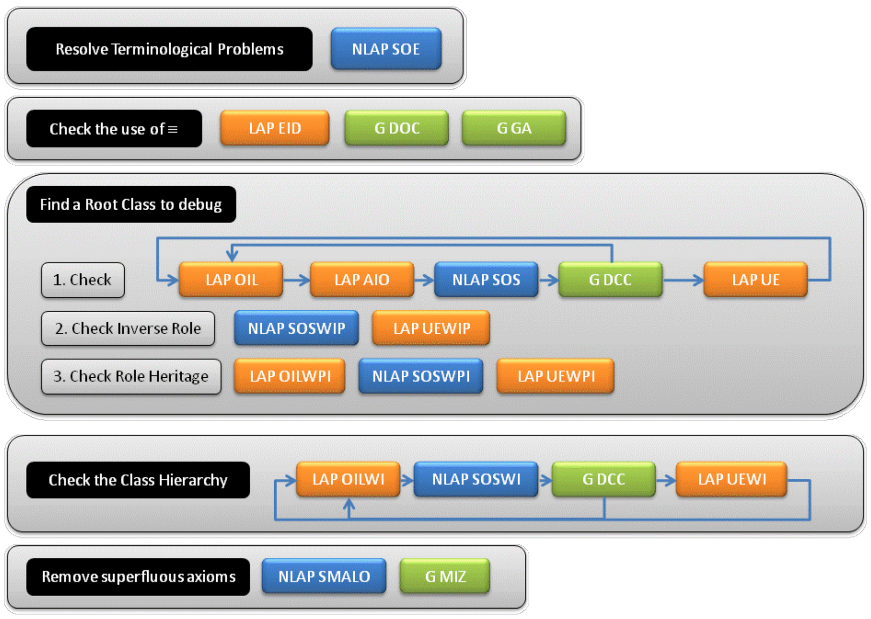

Figure 2: Detailed debugging strategy based on antipatterns

Finding appropriate recommendations at each step in the process may be a hard task, even for experienced ontology developers. Thus we also propose a more detailed strategy to be followed for obtaining recommendations, based on the catalogue of anti-patterns described in section 2. We propose to follow a specific order, based on our experience, as summarised in Figure 2.

First, we recommend solving terminological problems (SOE), checking the use of equivalence and disjoint constructors between classes (EID, DOC) and applying the guideline GA in order to make formal definitions easier to understand, grouping in the same definition all the axioms dealing with the same role. These are the easiest anti-patterns to detect and they are useful to clean other ontology definitions.

Then we propose checking the root unsatisfiable concepts in the ontology, using any off-the-shelf debugging tool (e.g., the justification system of Protégé). At that point the ontology developer can check the use of universal restrictions (OIL), existential restrictions (AIO, SOS, DCC) and their combinations (UE). Sometimes, inconsistencies arise from a combination of several anti-patterns, so that is the reason why there are loops in the figure.

After solving problems in root unsatisfiable classes, reasoners have to be used again to go down in the concept hierarchy, and apply iteratively the same steps, except for the case of patterns like OILWI and UEWI, where the process is from leaves to the root classes.

In the end we propose removing superfluous axioms to improve the clarity of the ontology (SMALO, MIZ)

3.2 Recommendations for Individual Anti-Patterns

In this section, for each anti-pattern we provide one or several recommendations that can be followed by ontology developers in order to remove inconsistencies (in the case of logical anti-patterns) or to improve the ontology structure (for NLAP and Guidelines).

3.2.1 Recommendations for Logical Anti-patterns

AntiPattern AIO

~~$C1 \subseteq \exists R.(C2 \cap C3)$,~~ disj(C2,C3) =>

#1 $\mathbf{C1 \subseteq \exists R.(C2 \cup C3)}$

#2 $\mathbf{C1 \subseteq \exists R.C2 \cap \exists R.C3}$

Replace the logical conjunction by the logical disjunction, or by the conjunction of two existential restrictions, which will lead into another non-logical anti-pattern.

AntiPattern OIL

~~$C1 \subseteq \forall R.C2$, $C1 \subseteq \forall R.C3$~~,disj(C2,C3) => $\mathbf{C1 \subseteq \forall R.(C2 \cup C3)}$

If it makes sense, the two universal restrictions could be transformed into only one that refers to the disjunction of C2 and C3.

AntiPattern OILWI

$C1 \subseteq C2$, $C1 \subseteq \forall R.C3$, ~~$C2 \subseteq \forall R.C4$~~, disj(C3,C4) =>

$\mathbf{C2 \subseteq \forall R.(C3 \cup C4)}$

If it makes sense, add the universal restriction of C1 to C2, using a class disjunction. As described in section 3.1, the strategy to be followed for correcting these antipatterns is to start from the leaves of the class hierarchy and check the parent class axiom dealing with the same role R. Normally, in case that the ontology developer wants to keep the parent definition, a new class (C2') has to be added with the parent definition as a sibling class of C1. By adding a new class, the ontology developer has to verify that the taxonomic relations between C2 and its subclasses are still up-to-date (disjoint and exhaustive decompositions, if they existed). Another check to be done is whether there is any difference between the formal definitions of some of the existing classes up in the hierarchy, since it could be the case that two upper level classes now have the same definition, what may mean that one of them is not useful anymore.

AntiPattern OILWPI

$R1 \subseteq R2$, ~~$C1 \subseteq \forall R1.C2$, $C1 \subseteq \forall R2.C3$~~, disj(C2,C3) => **$C1 \subseteq \forall R2.(C2 \cup C3)$**

If it makes sense, follow the same recommendation proposed for the OIL anti-pattern, as if the axiom $C1 \subseteq \forall R2.C2$ had been included.

AntiPattern UE

~~$C1 \subseteq \forall R.C2$, $C1 \subseteq \exists R.C3$~~, disj(C2,C3) =>

#1: **$C1 \subseteq \forall R.(C2 \cup C3)$**

#2: $C1 \subseteq \forall R.(C2 \cup C3) \cap \exists R.C2$

#3: $C1 \subseteq \forall R.(C2 \cup C3) \cap \exists R.(C2 \cup C3)$

#4: **$C1 \subseteq \exists R.(C2 \cup C3)$**

As described in [12], ontology developers sometimes do not distinguish clearly between existential and universal restrictions. In general, the solution #2 is the most accepted one. However, there are other possibilities that can be selected according to the intended meaning of the original definitions.

AntiPattern UEWI

(I) $C1 \subseteq C2$, $C1 \subseteq \exists R.C3$, ~~$C2 \subseteq \forall R.C4$~~, disj(C3,C4) => **$C1 \subseteq \forall R.(C3 \cup C4), C2 \subseteq \forall R.(C3 \cup C4)$**

(II) $C1 \subseteq C2$, ~~$C1 \subseteq \forall R.C3$~~, $C2 \subseteq \exists R.C4$, disj(C3,C4) => **$C1 \subseteq \forall R.(C3 \cup C4) \cap \exists R.C4$**

These two anti-patterns are very difficult to debug, since there could be multiple reasons for their appearance. Our proposal is aimed at obtaining consistency, but it should be clearly analysed by the ontology developer.

AntiPattern UEWPI

$R1 \subseteq R2$, $C1 \subseteq \exists R1.C2$, ~~$C1 \subseteq \forall R2.C3$~~, disj(C2,C3) => **$C1 \subseteq \forall R2.(C2 \cup C3)$**

If it makes sense, we propose following the UE recommendations after adding the restriction $C1 \subseteq \forall R2.C2$. Like in the previous case, there are several solutions for this antipattern. For instance, in some of our experiments we realised that our ontology developers misunderstood the subPropertyOf relation between roles, and hence the solution was to remove that relation.

AntiPattern UEWIP

$C2 \subseteq \exists R^{-1}.C1$, ~~$C1 \subseteq \forall R.C3$~~, disj(C2, C3)
$C1 \subseteq \forall R.(C2 \cup C3)$

We propose adding the reverse axiom of the C2 definition ($C1 \subseteq \exists R.C2$) and, if it makes sense, follow the UE recommendations.

AntiPattern EID

~~$C1 \equiv C2$, disj(C1,C2)~~ **=> $C1 \subseteq C2$ or C2 is a label of C1**.

The ontology developer should be asked about whether he really wants to define a synonym or a subclass-of relation. Maybe he made an error and one of these axioms can be removed. 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.

3.2.2 Recommendations for Non-Logical Anti-patterns

AntiPattern SOE

C1 ~~$\equiv C2$~~ => **C1.[rdfs:label|comment] = C2.[rdfs:label|comment]**

In this case, it is quite normal that C1 or C2 are not used in any other definitions of the ontology. Considering that this is the case for C2, the recommendation would be, if it makes sense, to add the label and natural language definition of C2 to the corresponding annotation properties of C1. If the two concepts are used in other ontology axioms, then the expert can decide which one to remove, if it makes sense, and the renaming has to be done across the whole ontology.

AntiPattern SOS

~~$C1 \subseteq \exists R.C2$, $C1 \subseteq \exists R.C3$~~, disj(C2,C3) => **$C1 \subseteq \exists R.(C2 \cup C3)$**

This anti-pattern has to be debugged taking into account the relationship between C2 and C3. If it had been $C2 \subseteq C3$ then the ontology developer may want to define C1 with the most specialized axiom ($C1 \subseteq \exists R.C2$) or the most general one ($C1 \subseteq \exists R.C3$). If C2 and C3 are disjoint, the expert has to be asked about whether this is really what he intended to represent, explaining clearly that an instance of C1 would need to have R connected to at least two other instances, each one coming from each of the concepts C2 and C3. Normally, the solution will be the one proposed, using a disjunction.

AntiPattern SOSWI

$C1 \subseteq C2$, $C1 \subseteq \exists R.C3$, ~~$C2 \subseteq \exists R.C4$~~, disj(C3,C4) =>
$C2 \subseteq \exists R.(C3 \cup C4)$

If it makes sense, add the existential restriction $\exists R.C3$ to the definition of class C2 and apply the SOS recommendation. The strategy for correcting this antipattern is similar to what was proposed for OILWI, that is, move from the leaves to the concepts that are up in the hierarchy.

AntiPattern SOSWPI

R1$\subseteq$R2, ~~C1$\subseteq\exists$R1.C2~~, C1$\subseteq\exists$R2.C3, disj(C2,C3) =>
C1$\subseteq\exists$R1.(C2$\cup$C3)

If it makes sense, add the axiom C1$\subseteq\exists$R2.C2 and follow the SOS recommendations.

AntiPattern SOSWIP

C2$\subseteq\exists R^{-1}$ .C1, ~~C1$\subseteq\exists$R.C3~~, disj(C2,C3) =>
C1$\subseteq\exists$R.(C2$\cup$C3)

If it makes sense, add the axiom C1$\subseteq\exists$R.C2 and follow the SOS recommendations.

AntiPattern SMALO

C1$\subseteq\exists$R.C2, ~~C1$\subseteq$($\geq$1 R.T)~~

The restriction ($\geq$1 R.T) can be removed, since it does not provide any additional knowledge.

3.2.3 Recommendations for Guidelines

The recommendations provided for these anti-patterns, given their characteristics, are mainly focusing on making the ontology easier to understand by ontology developers, and do not make any change with respect to the semantics or intended meaning of the ontology. We have determined that the proposed changes make the ontology easier to understand by asking a good range of ontology developers about their preferences when analysing ontologies that were not developed by them.

Guideline DOC

~~C1 ≡ not C2~~ => **disj(C1,C2)**

Guideline DCC

~~C1$\subseteq\exists$R.C2,~~ C1$\subseteq$(=2R.T) => **C1$\subseteq\forall$R.C2**

Guideline GA

~~C1$\subseteq\forall$R.C2~~, ~~C1$\subseteq$($\geq$2R.T)~~ => **C1$\subseteq\forall$R.C2$\cap$C1$\subseteq$($\geq$2R.T)**

Guideline MIZ

~~C1$\subseteq$($\geq$0R.T)~~

CONCLUSIONS AND FUTURE WORK

In this paper we have described a strategy for OWL ontology debugging that can be used in combination with existing OWL ontology debugging services in order to improve the efficiency of the debugging process by having a predefined set of suggested actions to be performed by ontology developers. We have obtained this strategy taking into account our experience in the development of DL-based ontologies and a careful analysis about how ontology developers and ontology engineers debug their ontologies nowadays.

As part of the work that we had to do in order to come up with this strategy, we have collected a list of common anti-patterns that can be found in ontologies and that cause a large percentage of the inconsistency problems. Besides, we have listed some anti-patterns that do not have an impact on the logical consequences of the ontology being developed, but that are of importance in order to reduce the number of errors in the intended meaning of ontologies or to improve their understandability.

For the time being, our strategy is mainly manual, where debugging tools are used to detect some of the logical antipatterns, although it remains to the user to find exactly where the antipattern is and which antipattern is applied. Hence as part of our future work we are aiming at implementing appropriate tools that can be combined with these tools and integrated in ontology engineering suites, providing a better support for the whole task of OWL ontology debugging.

We have initially tested our strategy with a set of inconsistent ontologies in the geographical domain developed by ontology developers. However, more systematic evaluations have to be done in the future to test the adequacy of the proposed strategy, the clarity of the provided recommendations, the time reduction in the debugging task and the exhaustiveness of the anti-patterns catalogue. We will perform these evaluations once that we have the first implementation of the support system. Besides, the list of anti-patterns could be used as an additional piece of information to be used to determine the overall quality of an ontology.

Another part of future work will be related to applying this strategy for the debugging of well-known inconsistent ontologies (e.g., TAMBIS, the testbed proposed by [14], etc.). In this case we would be extending our work to that of experts debugging ontologies that have not been written by them. And we will also focus on how anti-patterns are usually combined together and how more complex ones can be found that can speed up even more the debugging process.

Finally, some of the explanations that we have provided for the appearance of anti-patterns are related to the order in which some of the restrictions and axioms have been added to the ontology. Hence keeping a record of the changes that have been made to the ontology following well-known ontology change management systems could be useful in order to incorporate this into the anti-pattern detection phases and providing possible ranked solutions to them.

ACKNOWLEDGMENTS

This work was partially funded by the COST Action C21 sponsored by the European Commission under the grant number STSM-C21-04241.

REFERENCES

[1] Clark P, Thompson J, Porter B. “Knowledge patterns” In Proceedings of 7th International Conference *Principles of Knowledge Representation and Reasoning* (KR), Breckenridge, Colorado, USA: 591-600. (2000)

[2] Erich G; Helm R, Johnson R, Vlissides J. “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley. ISBN 0-201-63361-2. (1995)

[3] Gangemi A. “Design Patterns for Legal Ontology Construction”. In P. Casanovas, P. Noriega, D. Bourcier, F. Galindo (Ed.), *Trends in Legal Knowledge: The Semantic Web and the Regulation of Electronic Social Systems* European Press Academic Publishing. (2007).

[4]. Horridge M, Parsia B, Sattler U. “Laconic and Precise Justifications in OWL”. In Proceedings of the *7th International Semantic Web Conference* (ISWC), Karlsruhe, Germany; LNCS 5318: 323-338. (2008).

[5]. Kalyanpur A, Parsia B, Sirin E, Cuenca-Grau B. “Repairing Unsatisfiable Classes in OWL Ontologies”. In Proceedings of the *3rd European Semantic Web Conference* (ESWC), Budva, Montenegro; LNCS 4011: 170-184 (2006)

[6] Koenig A. “Patterns and Antipatterns”. *Journal of Object-Oriented Programming* 8, (1): 46–48. (1995).

[7] Collection of antipatterns available at http://wiki.loa-cnr.it/index.php/LoaWiki:MixedDomains

[8] Paslaru E, Simperl B, Tempich C, Sure Y. Ontocom:. “A cost estimation model for ontology engineering”. In Proceedings of the *5th International Semantic Web Conference* (ISWC), Athens, USA; LNCS 4273: 625-639. (2006)

[9] Presutti V, Gangemi A, David S, Aguado G, Suarez-Figueroa MC, Montiel E, Podeva M. “Neon Deliverable D2.5.1: A Library of Ontology Design Patterns” available at <http://www.neon-project.org>

[10] Presutti V., Gangemi A. “Content Ontology Design Patterns as practical building blocks for web ontologies”. In Proceedings of the 27th *International Conference on Conceptual Modeling* (ER), Barcelona, Spain, LNCS 5231: 128-141(2008).

[11] Rech J, Feldmann R L, Ras E. "Knowledge Patterns". In M. E. Jennex (Ed.), *Encyclopedia of Knowledge Management* (2nd Edition), IGI Global, USA, (2009).

[12] Rector AL, Drummond N, Horridge M, Rogers L, Knublauch H, Stevens R, Wang H, Wroe C. “OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns”. In Proceedings of the 14th *International Conference Knowledge Acquisition, Modeling and Management* (EKAW), Whittlebury Hall, UK. LNCS 3257: 63-81 (2004)

[13] Schlobach S, Cornet R. “Non-standard reasoning services for the debugging of description logic terminologies”. In Proceedings of the 18th *International Joint Conference on Artificial Intelligence* (IJCAI), Acapulco, Mexico: 355-362. (2003).

[14] Stuckenschmidt H. “Debugging OWL Ontologies – a Reality Check”. In Proceedings of the 6th *International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge* (EON-SWSC-2008), Tenerife, Spain. (2008).

[15] Vilches-Blázquez LM, Bernabé-Poveda MA, Suárez-Figueroa MC, Gómez-Pérez A, Rodríguez-Pascual AF *Towntology & hydrOntology: Relationship between Urban and Hydrographic Features in the Geographic Information Domain*. In: Ontologies for Urban Development. Studies in Computational Intelligence, vol. 61, Springer: 73–84. (2007)

[16] Wang, H., Horridge M, Rector A, Drummond N, Seidenberg J. *Debugging OWL-DL Ontologies: A heuristic approach*. In Proceedings of the *4th International Semantic Web Conference* (ISWC), Galway, Ireland; LNCS 3729: 745-757(2005)

[17] W3C. Semantic Web Best Practices and Deployment Working Group available at http://www.w3.org/2001/sw/BestPractices/