OnlynessIsLoneliness (OIL)
O. Corcho, C. Roussey

To cite this version:
OnlynessIsLoneliness (OIL)
http://ontologydesignpatterns.org/wiki/Submissions:
OnlynessIsLoneliness_(OIL)

Oscar Corcho\textsuperscript{1} and Catherine Roussey\textsuperscript{23}

\textsuperscript{1} Ontology Engineering Group, Departamento de Inteligencia Artificial, Universidad
Politecnica de Madrid, Spain
ocorcho@fi.upm.es,
\textsuperscript{2} Cemagref, 24 Av. des Landais, BP 50085, 63172 Aubiere, France
catherine.roussey@cemagref.fr,
\textsuperscript{3} Universite de Lyon, CNRS, Universite Lyon 1, LIRIS UMR5205,
Villeurbanne, France
catherine.roussey@liris.cnrs.fr,
WWW home page: http://liris.cnrs.fr/membres?idn=croussey

1 Introduction

Our work is based on the debugging process of 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, leading to unwanted unsatisfiable classes. Our patterns were
first found during the debugging process of a medium-sized OWL ontology (165
classes) developed by a domain expert in the area of hydrology [9]. The first
version of this ontology had a total of 114 unsatisfiable classes. The information
provided by the debugging systems used ([3], [5]) on (root) unsatisfiable classes
was not easily understandable by domain experts to find the reasons for their
unsatisfiability. And in several occasions during the debugging process the gen-
eration of justifications for unsatisfiability took several hours, what made these
tools hard to use, confirming the results described in [8]. Using this debugging
process and several other real ontologies debugging one, 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 definitions instead of correcting errors in their formalisations.

We have identified a set of patterns that are commonly used by domain
experts in their DL formalisations and OWL implementations, and that nor-

mally result in unsatisfiable classes or modelling errors ([1], [7]). Thus they are
antipatterns. [6] define antipatterns as patterns that appear obvious but are in-
effective or far from optimal in practice, representing worst practice about how
to structure and build software. We also have made an effort to identify common
alternatives for providing solutions to them, so that they can be used by domain
experts to debug their ontologies.
All these antipatterns come from a misuse and misunderstanding of DL expressions by ontology developers. Thus they are all Logical AntiPatterns (LAP): they are independent from a specific domain of interest, but dependent on the expressivity of the logical formalism used for the representation.

2 Pattern

2.1 Problem

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. Figure 1 illustrates this problem: grey squares represent instances of $C_2 \cap C_3$ that cannot exist. In general, this is because the ontology developer forgot the previous axiom in the same class or in any of the parent classes.

![Fig. 1. A graphical representation of OIL antipattern.](image)

\[ C_1 \sqsubseteq \forall R. (C_2); C_1 \sqsubseteq \forall R. (C_3); \text{Disj}(C_2, C_3); \]

Notice that to be detectable, $R$ property must have at least a value, normally specified as a (minimum) cardinality restriction for that class, or with existential restrictions.

Covers Requirements When this antipattern appears during the debugging process, you have to first explain to the domain expert the meaning of this formalisation using a schema like the one of the Figure 1. Then you could ask

\[ \text{This does not mean that the ontology developer has explicitly expressed that } C_2 \text{ and } C_3 \text{ are disjoint, but that these two concepts are determined as disjoint from each other by a reasoner. We use this notation as a shorthand for } C_2 \cap C_3 \sqsubseteq \bot. \]
him some questions to find out where is the problem. For example, you could ask:
- Should $C_1$ be linked with the $R$ property to $C_2$?
- Should $C_1$ be linked with the $R$ property to $C_3$?
- Does $C_1$ have to be linked only to $C_2$ with the $R$ property?
- Does $C_1$ have to be linked only to $C_3$ with the $R$ property?
- Are you sure that $C_2$ and $C_3$ are disjoint?

2.2 Solution
If it makes sense, we propose the domain expert to transform the two universal
restrictions into only one that refers to the logical disjunction of $C_2$ and $C_3$.
Another alternative solution, which is used by most part of automatic debugging
tool is to remove one of the axioms.

\[ C_1 \sqsubseteq \forall R.C_2; C_1 \sqsubseteq \forall R.C_3; \text{Disj}(C_2, C_3); \Rightarrow C_1 \sqsubseteq \forall R.(C_2 \sqcup C_3); \]

2.3 Example
The following section describes two definitions from HydrOntology where this
antipattern can be found and their English translations. Notice that in each
example, the antipattern corresponds to a part of the class definition.

Example Problem about Transitional Water
\[ \text{Agua de Transición} \sqsubseteq \forall \text{está próxima.} \text{Agua Marina} \sqcap \forall \text{está próxima. Desembocadura} \sqcap \text{1 está próxima.} \top; \]
\[ \text{Transitional Water} \sqsubseteq \forall \text{vis nearby.} \text{Sea Water} \sqcap \forall \text{vis nearby.} \text{River Mouth} \sqcap \text{1 is nearby.} \top; \]

Example Solution about Transitional Water
\[ \text{Agua de Transición} \sqsubseteq \forall \text{está próxima.} (\text{Agua Marina} \sqcup \text{Desembocadura}) \sqcap \text{1 está próxima.} \top; \]
\[ \text{Transitional Water} \sqsubseteq \forall \text{vis nearby.} (\text{Sea Water} \sqcup \text{River Mouth}) \sqcap \text{1 is nearby.} \top; \]

Example Problem about Wet Zone
\[ \text{Zona Humeda} \sqsubseteq \forall \text{Humedal} \sqcap \forall \text{es inundada.} \text{Agua Marina} \sqcap \forall \text{es inundada.} \text{Agua Superficial} \sqcap \text{1 es inundada.} \top; \]
\[ \text{Wet Zone} \sqsubseteq \forall \text{Wetlands} \sqcap \forall \text{are inundated.} \text{Sea Water} \sqcap \forall \text{are inundated.} \text{Surface Water} \sqcap \text{1 are inundated.} \top; \]

Example Solution about Wet Zone
\[ \text{Zona Humeda} \sqsubseteq \forall \text{Humedal} \sqcap \forall \text{es inundada.} (\text{Agua Marina} \sqcup \text{Agua Superficial}) \sqcap \text{1 es inundada.} \top; \]
\[ \text{Wet Zone} \sqsubseteq \forall \text{Wetlands} \sqcap \forall \text{are inundated.} (\text{Sea Water} \sqcup \text{Surface Water}) \sqcap \text{1 are inundated.} \top; \]
2.4 Related Resources and Pattern Usage

All the information related to the debugging of the Hydrontology ontology can be found in urlhttp://www.dia.fi.upm.es/ocorcho/OWLDebugging/. The debugging strategy using this antipattern is described in [2]. Other antipatterns found during the debugging task are defined in [1] and [7].

3 Summary and Future Work

This antipattern can be found in ontologies and may cause inconsistency problems. We provide a solution to it, so that it can be used by domain experts to debug their ontologies. In the future, we aim at implementing additional tools to help in the identification of antipatterns in well-known inconsistent ontologies (e.g., TAMBIS). For the time being we have started applying the OPPL language [4] for this task, with promising results.

References