OnlynessIsLoneliness (OIL)
O. Corcho, C. Roussey

To cite this version:

HAL Id: hal-00473250
https://hal.archives-ouvertes.fr/hal-00473250

Submitted on 14 Apr 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
OnlynessIsLoneliness (OIL)

http://ontologydesignpatterns.org/wiki/Submissions:
OnlynessIsLoneliness_(OIL)

Oscar Corcho\textsuperscript{1} and Catherine Roussey\textsuperscript{2,3}

\textsuperscript{1} Ontology Engineering Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es,

\textsuperscript{2} Cemagref, 24 Av. des Landais, BP 50085, 63172 Aubière, France
    catherine.roussey@cemagref.fr,

\textsuperscript{3} Université de Lyon, CNRS, Université 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 generation 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 normally result in unsatisfiable classes or modelling errors ([1], [7]). Thus they are antipatterns, [6] define antipatterns as patterns that appear obvious but are ineffective 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.

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

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

4 This does not mean that the ontology developer has explicitly expressed that $C_2$ and $C_3$ 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 $C_2 \cap C_3 \sqsubseteq \bot$. 

---

Fig. 1. A graphical representation of OIL antipattern.
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 ⊑ ∀R.C_2; C_1 ⊑ ∀R.C_3; Disj(C_2, C_3); ⇒ C_1 ⊑ ∀R.(C_2 ∪ 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

```
Aguas_de_Transición ⊑ ∀está_próxima.Aguas_Marinas ⊓
∀está_próxima.Desembocadura ⊓ = 1está_próxima.⊤;

Transitional_Water ⊑ ∀vis_nearby.Sea_Water ⊓ ∀vis_nearby.River_Mouth ⊓
= 1is_nearby.⊤;
```

Example Solution about Transitional Water

```
Aguas_de_Transición ⊑ ∀está_próxima.(Aguas_Marinas ⊓ Desembocadura) ⊓
= 1está_próxima.⊤;

Transitional_Water ⊑ ∀vis_nearby.(Sea_Water ⊓ River_Mouth) ⊓
= 1is_nearby.⊤;
```

Example Problem about Wet Zone

```
Zona_Humeda ⊑ ∀Humedal ⊓ ∀es_inundada.Aguas_Marinas ⊓
∀es_inundada.Aguas_Superficiales ⊓ ≥ 1es_inundada.⊤;

Wet_Zone ⊑ ∀Wetlands ⊓ ∀are_inundated.Sea_Water ⊓
∀are_inundated.Surface_Water ⊓ ≥ 1are_inundated.⊤;
```

Example Solution about Wet Zone

```
Zona_Humeda ⊑ ∀Humedal ⊓
∀es_inundada.(Aguas_Marinas ⊓ Aguas_Superficiales) ⊓ ≥ 1es_inundada.⊤;

Wet_Zone ⊑ ∀Wetlands ⊓ ∀are_inundated.(Sea_Water ⊓ Surface_Water) ⊓
≥ 1are_inundated.⊤;
```
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