Ontological Anti-Patterns in Taxonomic Structures

Tiago Prince Sales[1] **and Giancarlo Guizzardi**[2]

[1]ISTC-CNR Laboratory for Applied Ontology, Trento, Italy

[2]Conceptual and Cognitive Modelling Research Group (CORE), Free University of Bozen-Bolzano, Bolzano, Italy

tiago.princesales@unitn.it, giancarlo.guizzardi@unibz.it

Abstract. *Over the years, there is a growing interest in employing theories from philosophical ontology, cognitive science and linguistics to devise theoretical, methodological and computational tools for conceptual modeling, knowledge representation and domain ontology engineering. In this paper, we discuss one particular kind of such tools, namely, ontological anti-patterns. Ontological anti-patterns are error-problem modeling structures that can create a deviation between the possible and the intended interpretations of an ontology. The contributions of the paper are three-fold. Firstly, we propose some empirically elicited ontological anti-patterns related to the modeling of taxonomic structures. Secondly, we advance a series of rectification plans that can be used to eliminate the occurrence of these anti-patterns in domain ontologies. Finally, we present a model-based computational tool that supports the automated detection, analysis, and elimination of these anti-patterns.*

1. Introduction

In recent years, there has been an increasing interest in the application of ontologies in conceptual modeling, knowledge representation and domain engineering. This includes the use of foundational ontologies to improve the theory and practice of these disciplines [Almeida et al. 2009, Guizzardi 2014]. In these scenarios, foundational ontologies play a key role in improving the conceptual quality of models by supporting communication, problem-solving, meaning negotiation and, chiefly, semantic interoperability in its various manifestations (e.g., enterprise and database application integration) [Nardi et al. 2013a].

Given the increasing complexity and criticality of domains in which ontologies are being developed (e.g., finances, life sciences, public safety, convergence networks), there is an urging need for developing a new generation of complexity management tools for engineering these artifacts [Guizzardi 2014]. These include a number of methodological and computational tools that are grounded on sound theoretical foundations. In particular, as defended in [Guizzardi 2014], we should advance in conceptual modeling, in general, and in ontology engineering, in particular, a well-tested body of knowledge in terms of Ontology Patterns, Ontology Pattern Languages and Ontological Anti-Patterns. This article focuses on the latter.

An anti-pattern is a recurrent error-prone modeling decision [Koenig 1995]. In this paper, we are interested in one specific sort of anti-patterns, namely, model structures that, albeit producing syntactically valid conceptual models, are prone to result in unintended domain representations. In other words, we are interested in configurations that, when used in a model, will typically cause the set of valid (possible) instances of that model to differ from the set of instances representing intended state of affairs in that domain [Guizzardi 2005]. Such a difference occurs either because the model allows unintended model instances or because it forbids intended ones. We name these configurations Ontological Anti-Patterns.

In this article, we focus on the study of Ontological Anti-Patterns in a particular modeling language named OntoUML [Guizzardi 2005]. OntoUML is a language whose meta-model has been designed to comply with the ontological distinctions and axiomatization of a theoretically well-grounded foundational ontology named UFO (Unified Foundational Ontology) [Guizzardi 2005, Guizzardi et al. 2015]. UFO is an axiomatic formal theory based on theories from Formal Ontology in Philosophy, Philosophical Logics, Cognitive Psychology and Linguistics. OntoUML has been successfully employed in several industrial projects in different domains, such as petroleum and gas, digital journalism, complex digital media management, off-shore software engineering, telecommunications, retail product recommendation, and government [Guizzardi et al. 2015]. A recent study shows that UFO is the second-most used foundational ontology in conceptual modeling and the one with the fastest adoption rate [Verdonck and Gailly 2016]. Moreover, the study also shows OntoUML is among the most used languages in ontology-driven conceptual modeling (together with UML, (E)ER, OWL and BPMN).

This article can be seen as complementary to our earlier work on anti-patterns. In [Sales and Guizzardi 2015], we have focused on anti-patterns that are connected to the modeling of material relations (roughly domain associations), in [Sales and Guizzardi 2016], on those connected to the modeling of roles, and in [Sales and Guizzardi 2017], on those emerging from modeling part-whole relations. Now we focus on anti-patterns that emerge when modeling certain taxonomic structures.

The contributions of this paper are three-fold. Firstly, we contribute to the identification of three new Ontological Anti-Patterns for conceptual modeling, in general, and for OntoUML, in particular. Secondly, after precisely characterizing these anti-patterns, we propose a set of refactoring plans that can be adopted to eliminate the possible unintended consequences induced by the presence of each of these anti-patterns. Finally, we present an extension for the Menthor Editor, an open-source OntoUML model-based editor that: (i) automatically detects anti-patterns in their models; (ii) supports users in exploring whether the presence of an anti-pattern indeed characterizes a modeling error; (iii) automatically executes refactoring plans to rectify the model.

The remainder of this article is organized as follows: in Section 2, we briefly elaborate on the modeling language OntoUML and some of its underlying ontological categories, with a particular focus on the modeling of taxonomic structures; In Section 3, we first briefly present the anti-pattern elicitation method employed here and characterizes the model benchmark used in this research; In Section 4, we present the newly elicited Ontological Anti-Patterns with their unintended consequences, as well as possible solutions for their rectification in terms of model refactoring plans; Section 5 elaborates on the extensions implemented in the OntoUML editor taking into account these anti-patterns; Finally, Section 6 presents some final considerations.

2. Background: A Brief introduction to UFO and OntoUML

OntoUML is a language whose meta-model has been designed to comply with the ontological distinctions and axiomatization of a theoretically well-grounded foundational ontology named UFO (Unified Foundational Ontology) [Guizzardi 2005, Guizzardi et al. 2015]. In the remainder of this section, we briefly explain a selected subset of the ontological distinctions put forth by the Unified Foundational Ontology (UFO). We also show how these distinctions are represented by the modeling primitives of OntoUML.

Take a domain in reality restricted to *endurants* [Guizzardi 2005] (as opposed to events or occurrents). Central to this domain we will have a number of object *Kinds*, i.e., the genuine fundamental types of objects that exist in this domain. The term "kind" is meant here in a strong technical sense, i.e., by a kind, we mean a type capturing *essential* properties of the things it classifies. In other words, the objects classified by that kind could not possibly exist without being of that specific kind.

Kinds tessellate the possible space of objects in that domain, i.e., all objects belong necessarily to exactly one kind. However, we can have other static subdivisions (or subtypes) of a kind. These are naturally termed *Subkinds*. As an example, the kind 'Person' can be specialized in the subkinds 'Man' and 'Woman'.

Object kinds and subkinds represent essential properties of objects (they are also termed *rigid or static types* [Guizzardi 2005]). We have, however, types that represent contingent or *accidental* properties of objects (termed *anti-rigid types* [Guizzardi 2005]). These include *Phases* and *Roles*. The difference between the contingent properties represented by a phase and a role is the following: phases represent properties that are intrinsic to entities; roles, in contrast, represent properties that entities have in a relational context, i.e., contingent relational properties.

Phases but also typically subkinds appear in OntoUML models forming (disjoint and complete, i.e., exhaustive) *generalization sets* (partitions) following a *Dividing Principle*. For example, we can have the following *phase partitions*: the one including 'Living Person' and 'Deceased Person' (as phases of 'Person' and according to a 'life status' dividing principle). Since they are exclusively composed of phases, these are all dynamic partitions [Guizzardi 2005]. To use a previously mentioned example, we can also have a (static) *subkind partition* formed by the subkinds 'Man' and 'Woman', dividing 'Person' according to 'gender'.

Kinds, Subkinds, Phases, and Roles are categories of object *Sortals*. In the philosophical literature, a sortal is a type that provides a *uniform principle of identity*, persistence, and individuation for its instances [Guizzardi 2005]. To put it simply, a sortal is either a kind (e.g., 'Person') or a specialization of a kind (e.g., 'Husband', 'Teenager', 'Woman'), i.e., it is either a type representing the essence of what things are or a subclassification applied to the entities that "have that same type of essence".

In contrast with sortals, types that represent properties shared by entities of multiple kinds are termed *Non-Sortals*. In UFO, we have three other types of non-sortals, namely *Categories*, *Mixins*, and *RoleMixins*. Categories represent necessary properties that are shared by entities of multiple kinds (e.g., the category 'Physical Object' represent properties of all kinds of entities that have masses, spatial extensions, etc.). In contrast, mixins represent shared properties that are necessary to some of its instances but accidental to others. For example, suppose we have the mixin ‘Physical Object’ capturing properties (e.g., having a ‘Weight’) that are necessary to ‘Cars’, while being accidental to instances of ‘Person’ (people are only physical objects when they instantiate the phase ‘Living Person’). Finally, RoleMixins are role-like types that can be played by entities of multiple kinds. An example is the role ‘Customer’ (which can be played by both people and organizations). Categories and mixins are, in contrast to rolemixins, considered as Relationally Independent Non-Sortals.

3. Methods and Materials

We identified the anti-patterns presented in this paper through an empirical and qualitative method. First, we started by assembling a repository of domain and core ontologies to validate. Then, for each ontology, we would select relevant fragments to inspect. For each fragment, we would search for potential issues using an approach called Visual Model Simulation [Sales and Guizzardi 2015]. This approach consists in: (i) converting OntoUML models into Alloy [Jackson 2012] specifications; (ii) generating possible model instances and contrasting these instances with the set of intended instances[1] of the model. Upon the identification of a mismatch, we would register it as a potential issue and identify in the model which structures (i.e., combination of language constructs) caused it. In the sequence, in order to define whether the identified structure is indeed problematic, we would either interact with the modelers (when available) or inspect the documentation accompanying the model. When we confirmed that there was indeed a problem, we would propose a possible solution to rectify the model and register it as a problem-solution pair. With the recently rectified model, we go back to step three. This iteration would be repeated until no more problems could be identified in the fragment and then, we would select another fragment to inspect. The analysis of a model stops whenever we inspect all of its relevant fragments. After inspecting each model, we analyze the generated problem-solution pairs in order to generalize them into pairs of anti-patterns and refactoring plans.

We systematically repeated the process we just described for the 54 ontologies available in the OntoUML model repository[2]. Out of these 54 models, 11 were developed as a part of academic research, such as O3 [Pereira and Almeida 2014], an ontology about organizational structures; 7 were developed in collaboration with private or public organizations, e.g. the MGIC Ontology [Bastos et al. 2011], a model developed for a regulatory agency responsible for administrating ground transportation services in Brazil. Moreover, 32 were designed as course assignments in post-graduate courses on ontology engineering, whilst the remainder 4 were developed in other contexts.

The ontologies we analyzed were created for a variety of purposes: 10 were designed to serve as core ontologies (e.g. UFO-S [Nardi et al. 2013b] for the domain of services); 10 were developed as means for an ontological analysis of existing formalizations, databases or modeling languages; 8 were designed to support the development of knowledge-based applications; 6 to support semantic interoperability between systems and/or organizations; 2 for enterprise modeling; and 26 of them for other purposes.

[1]The set of intended instances correspond to those that represent intended state of affairs [Guizzardi 2005] according the creators of the models.

[2]Most of these models are available at `http://www.menthor.net/model-repository.html`. The missing cases are due to non-disclosure agreements.

Moreover, the ontologies we analyzed were developed by modelers with varying levels of expertise in OntoUML: 22 were developed by beginners, whilst 32 were developed by experienced modelers. Finally, regarding the number of participants involved in the ontology construction, 35 models were developed individually, 15 were the product of a collaboration between 2-4 people, and 4 of them involved 7-10 people.

4. Ontological Antipatterns

4.1. Mixin with a Uniform Identity Principle (MixIden)

The anti-pattern named *Mixin with a Uniform Identity Principle (MixIden)* is motivated by the use of a non-sortal metaclass to characterize a type whose instances adhere to a single identity principle. Structurally, it corresponds to a type characterized as a mixin type (from now on, simply $Mixin$), i.e., a type stereotyped as either ≪mixin≫, ≪category≫, or ≪roleMixin≫), but whose direct subtypes ($Type_i$) are all sortals types (i.e., types stereotyped as ≪subkind≫, ≪role≫, ≪phase≫, ≪kind≫, ≪collective≫, or ≪quantity≫) that share a common identity provider ($IdProvider$) as an ancestor.

We say that the subtypes share an identity provider if: (i) there is exactly one identity provider amongst the subtypes and the remainder specialize it; or (ii) there is no identity providers amongst the subtypes, but all of them are directly or indirectly specializations of the same identity provider.

To unveil whether an *MixIden* occurrence is indeed an error, one should first assess if $Mixin$ represents a type can really possibly classify instances of different kinds, i.e., that obey multiple identity principles. If that is not the case, such type should be represented as a sortal, which can be achieved by: choosing a suitable sortal metaclass for for $Mixin$ (e.g. ≪subkind≫, ≪role≫ or ≪phase≫); make it specialize $IdProvider$ (see refactoring plan 1 in Table 1).

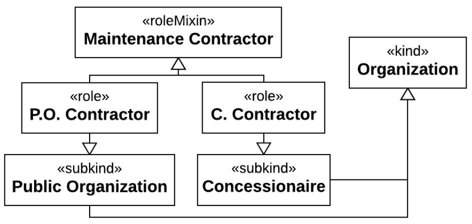
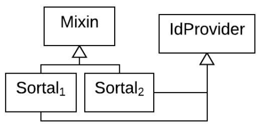
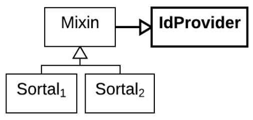
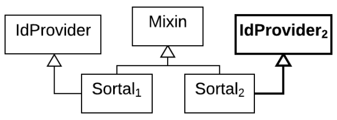
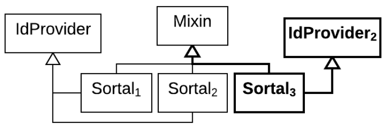


Figura 1. Adapted fragment from the MGIC Ontology exemplifying *MixIden*.

Conversely, if $Mixin$ indeed allows different identity principles for its instances, we recommended to make this explicit either by: (i) changing the identity provider of one of the current subtypes ($Type_i$); or (ii) specializing $Mixin$ with a new sortal that follows a different identity principle. These two alternatives are respectively depicted in refactoring plans 2 and 3 on Table 1.

To exemplify the *MixIden* anti-pattern, we discuss an example found in the MGIC ontology [Bastos et al. 2011], a reference model developed for the Brazilian Ground Transportation Agency. A fragment of this ontology, related to the domain of highway concessions, is depicted in Fig. 1. This fragment depicts two relevant types of organizations, namely concessionaires and public organizations. The former represents organizations that are created with the exclusive purpose of administrating federal highways, whilst the latter accounts for a general concept of publicly controlled organizations, such as regulatory agencies, ministries, and public companies. The focus of the fragment, however, is on *maintenance contractor*, a role played by organizations when they are responsible for maintenance works on any infrastructural components of a highway. According to Brazilian regulations, this role might only be played by public organizations and concessionaires, hence, the two "sub-roles"Public Organization Contractor (P.O. Contractor) and Concessionaire Contractor (C. Contractor).

Tabela 1. Summary of the *MixIden* anti-pattern.

Name (Acronym)		**Description**
Mixin with a Uniform Identity Principle (MixIden)		A non-sortal class specialized only by sortal types that obey a single identity principle, i.e. specialize a common identity provider class.
Pattern Roles		
Mult.	**Name**	**Allowed Metaclasses**
1	$Mixin$	≪mixin≫, ≪category≫, ≪roleMixin≫
1..*	$Sortal_i$	≪subkind≫, ≪role≫, ≪phase≫, ≪kind≫, ≪collective≫, ≪quantity≫
1..*	$IdProvider$	≪kind≫, ≪collective≫, ≪quantity≫
Generic Example		
		
Refactoring Plans		
1. Mixin to Sortal: change the stereotype of $Mixin$ to either ≪subkind≫, ≪role≫ or ≪phase≫ and make it specialize $IdProvider$. 		
2. Different Identity: set the identity provider of at least one of the $Sortal_i$ types to another class ($IdProvider_2$). 		
3. Missing sortal: specialize $Mixin$ with another sortal ($Sortal_3$) that follows a different identity principle (from $IdProvider_2$). 		

The reason why this fragment exemplifies a *MixIden* occurrence is the representation of Maintenance Contractor as a ≪roleMixin≫, even though all its subtypes, namely P.O. Contractor and C. Contractor, inherit their identity principles from the same supertype, the class Organization. As such, this fragment identifies an ontological mistake because only organizations can be contractors, and thus, the contractor role should be represented as a sortal class – following refactoring plan 1.

4.2. Mixin with Uniform Rigidity (MixRig)

The *Mixin with Uniform Identity (MixRig)* anti-pattern warns modelers about a potential misuse of the semi-rigidity meta-property to characterize a class. It does so by highlighting a ≪mixin≫ class (which we refer from now on as the $Mixin$) whose subtypes ($Type_i$) are either all rigid (≪subkind≫, ≪kind≫, ≪collective≫, ≪quantity≫, or ≪category≫) or all anti-rigid (≪role≫, ≪phase≫, ≪roleMixin≫).

To discover whether a *MixRig* occurrence is indeed a modeling mistakes, one should start by revisiting the semantics underlying the supposed semi-rigid universal. Is it truly the case that some of the instances of this universal necessarily instantiate it, whilst others do so accidentally? If the answer is no, this universal is not semi-rigid, and thus one should rectify the model by choosing a more suitable metaclass for $Mixin$ (see refactoring plan 1 in Table 2). If all the subtypes are rigid, this can be achieved by changing its stereotype to ≪category≫, and if they are all anti-rigid, by changing its stereotype to ≪roleMixin≫.

Contrarily, if $Mixin$ is indeed a semi-rigid type, the problem lies on its subtypes. A possibility is that one of these subtypes has been modeled using a metaclass that embeds an erroneous modal meta-property property (e.g., mistakenly as a ≪subkind≫ something should be in fact a ≪role≫). In this case, the solution is simply identifying which subtype carries was mistakenly modeled and fix it accordingly. Another possibility is that $Mixin$ has a subtype that is currently missing in the model, a type which would be characterized with a different rigidity property (e.g., a model fragment in which the present subtypes are subkinds, but in which there is non-represented role type). In this case, the modeler should explicitly include the missing subtype in the model. These two scenarios are described in the refactoring plans 2 and 3 in Table 2.

We illustrate an occurrence of *MixRig* in Fig. 2 with another taxonomic fragment extracted from the MGIC ontology [Bastos et al. 2011], but now in the domain of railway concessions. This fragment focuses on two types of assets that compose a railway infrastructure, namely stations and railway lots. Stations are those well-known buildings where trains stop for passengers to board and disembark, whilst railway lots are delimited pieces of land that compose the railway infrastructure, such as the areas alongside train tracks.

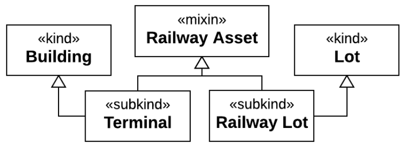
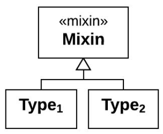
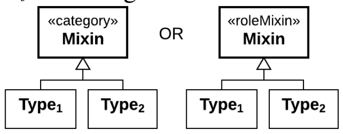
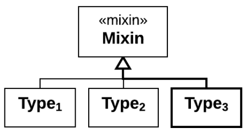
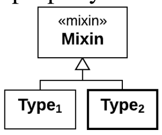


Figura 2. Adapted fragment from the MGIC Ontology exemplifying *MixRig*.

These fragment fits the *MixRig* anti-pattern because Railway Asset is represented as a ≪mixin≫ and its direct subtypes, Station and Railway Lot, are both subkinds, i.e., rigid types. Interacting with the authors of this model, we discovered that terminals always compose railway infrastructures, but lots (empty land areas) are accidental parts of this infrastructure. Thus, Railway Asset was properly characterized as a semi-rigid class, but Railway Lot was mistakenly represented as ≪subkind≫, for it should have been a ≪role≫ played by a lot when composing a Railway infrastructure.

Tabela 2. Summary of the *MixRig* anti-pattern.

Name (Acronym)		**Description**
Mixin with Uniform Rigidity (MixRig))		A ≪mixin≫ class specialized by classes with the same rigidity meta-property, i.e. either all rigid or all anti-rigid.
Pattern Roles		
Mult.	**Name**	**Allowed Metaclasses**
1	$Mixin$	≪mixin≫
1..*	$Type_i$	All class stereotypes except ≪mixin≫
Generic Example		
		
Refactoring Plans		
1. Not semi-rigid: change the stereotype of $Mixin$ either to a ≪category≫, if every $Type_i$ is rigid, or to a ≪roleMixin≫, if every $Type_i$ is anti-rigid.  «category» Mixin; OR; «roleMixin» Mixin; Type₁; Type₂; Type₁; Type₂		
2. Missing subtype: specialize $Mixin$ with a type ($Type_3$) that has a rigidity meta-property different from all $Type_i$. 		
3. Fix $Type_i$ stereotype: change the stereotype of at least one $Type_i$ (but not all of them) such that the new stereotype embeds a rigidity meta-property different from the remainder $Type_i$. 		

4.3. Generalization Set with a Heterogeneous Rigidity

The *Generalization Set With Mixed Rigidity (GSRig)* anti-pattern aims to identify model fragments that suggest the use of multiple specialization criteria within the same generalization set. A GSRig occurrence can be characterized by a generalization set whose common supertype is rigid (i.e. stereotyped as ≪kind≫, ≪quantity≫, ≪collective≫, ≪subkind≫ or ≪category≫) and that aggregates generalizations coming from both rigid and anti-rigid types (stereotyped as ≪phase≫, ≪role≫ and ≪roleMixin≫).

A *GSRig* occurrence can even lead to a logical contradiction, when it contains exactly one anti-rigid type. Such a contradiction arises from the combination of: (i) the rigidity constraints of the subtypes; (ii) the disjointness constraint of the set, which forbids instances of the rigid subtypes to simultaneously instantiate the anti-rigid type; and (iii) the completeness constraint of the set, which requires that every instance of the general type instantiate one of the subtypes. In this case, the only way for an individual to instantiate the anti-rigid subtype at hand would be by doing so since its creation. However, by definition of an anti-rigid type, it is necessary that such an individual possibly ceases to instantiate that type. This is, however, not possible in this case due to the rigidity of the complementary subtypes in that complete generalization set. In summary, this configuration forces the anti-rigid type at hand to classify its instances necessarily, i.e., to be in fact a rigid type.

To determine whether a *GSRig* occurrence indeed characterizes a modeling error, one should start by double checking the rigidity of the subtypes and fixing them accordingly if there are any mistakes. If after this step all subtypes are either rigid or anti-rigid, the model has been rectified. If that is not the case, one should revisit the common supertype's intended semantics to assess whether it should be characterized as a ≪mixin≫, a metaclass that exactly captures common properties shared by rigid and anti-rigid subtypes.

In case the aforementioned steps cannot be used to rectify the model, we suggest modelers to consider the specialization criterion used for each subtype. As we discussed in Section 2, a specialization criteria identifies the "dimension" considered for creating a subtype. For instance, the dimension used to differentiate the classes Child, Adult and Elder is the value of the age quality defined for the class Person. We uncovered throughout our analysis that when multiple dimensions were used to define subtypes within the same generalization set, it was likely for the model to have a problem. Thus, if that is the case, we suggest modelers to: (i) split the generalization set, if the constraints imposed by it are not accurate; or (ii) add a new rigid type as a sibling of the other rigid subtypes and make all anti-rigid subtypes in the generalization set specialize it.

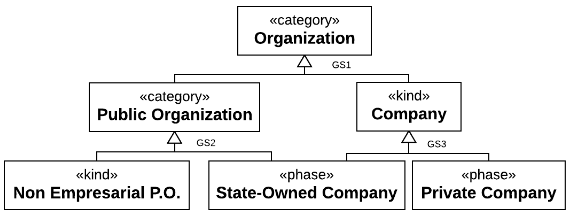


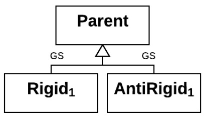
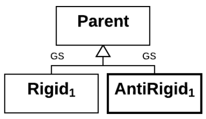
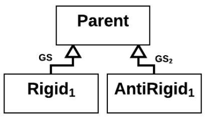
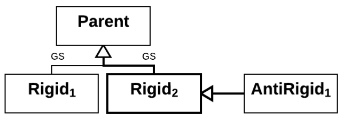
Figura 3. Adapted fragment from the MPOG Ontology exemplifying *GSRig*.

We exemplify *GSRig* by means of a fragment of a conceptual model on organizational structures published by the Brazilian Ministry of Planning, Budgeting and Management [3]. Fig. 3 depicts the top six types in the model's taxonomic structure. The root class, Organization, classifies all social entities according to the Brazilian law. Its first refinement distinguishes organizations according to their main activities - public organizations (owned by the government) and companies (owned by the private sector) arise, the former being further refined in non-empresarial public organizations (Non Empresarial P.O.), such as a ministry or a regulatory agency, and state-owned companies, such as

[3]Original name: Ministério do Planejamento, Orçamento e Gestão (MPOG)

the “Banco do Brasil” (a government owned bank).

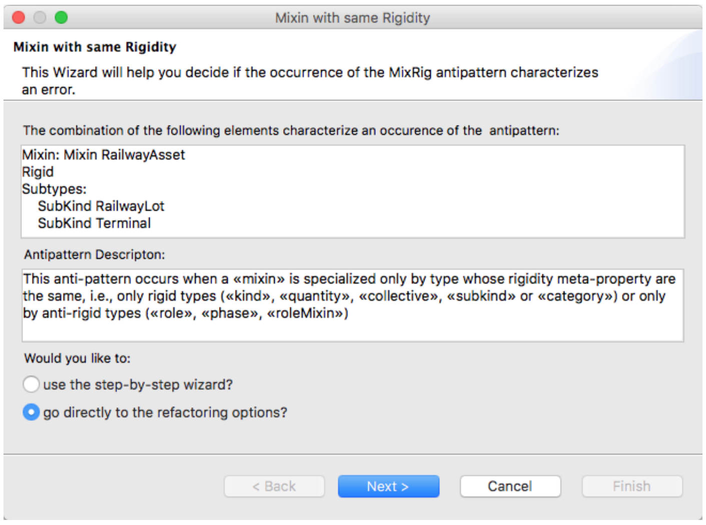
Tabela 3. Summary of the *GSRig* anti-pattern.

Name (Acronym)		**Description**
Generalization Set with Mixed Rigidity (GSRig))		A generalization set aggregating rigid and anti-rigid classes into a common rigid super-type.
Pattern Roles		
Mult.	**Name**	**Allowed Metaclasses**
1	GS	GeneralizationSet
1	$Parent$	≪kind≫, ≪collective≫, ≪quantity≫, ≪subkind≫, ≪category≫
1..*	$Rigid_i$	≪kind≫, ≪collective≫, ≪quantity≫, ≪subkind≫, ≪category≫
1..*	$AntiRigid_i$	≪role≫, ≪phase≫, ≪roleMixin≫
Generic Example		
		
Refactoring Plans		
1. All subtypes are rigid: transform every $AntiRigid_i$ into a rigid class by choosing suitable meta-classes.  **2. Missing generalization set:** add an extra generalization set (GS_2) to group all $AntiRigid_i$ classes.  **3. Hidden rigid type:** add a rigid type ($Rigid_2$) that specializes $Parent$ and generalizes every $AntiRigid_i$ in the generalization set. 		

The generalization set GS2 is the source of this *GSRig* occurrence. It aggregates the generalization relations coming from State-Owned Company (an anti-rigid type) and Non Empresarial P.O. (a rigid type) to Public Organization (also a rigid type). In fact, this example fits the particular case we previously discussed of a single anti-rigid type (the ≪phase≫ State-Owned Company) in a complete generalization set where all of its complementary types (in this case, only Non Empresarial Public Organization) are rigid. From the description of domain, we can clearly conclude that being a public organization, in the sense of being owned by the government, is a necessary condition for some organizations (e.g. the Ministry of Science and Technology will never be private owned) and contingent condition for others (e.g. a public company that is privatized).

5. Tool Support

In order to help modelers easily reuse the anti-patterns discussed in this paper to validate their ontologies, we implemented an anti-pattern management feature in Menthor Editor [4], an open-source ontology-driven conceptual modeling environment. Following the strategy adopted in [Sales and Guizzardi 2015], the anti-pattern management feature includes: (i) automatic anti-pattern detection based on the structures defined for each anti-pattern; (ii) a wizard-like component (depicted on Fig. 4) to support modelers in discovering whether an anti-pattern occurrence is an errors, and if so, what should be done to rectify it ; and (iii) automatic rectification of the models based on the refactoring plans presented in the previous section.


Mixin with same Rigidity

Mixin with same Rigidity

This Wizard will help you decide if the occurrence of the MixRig antipattern characterizes an error.

The combination of the following elements characterize an occurence of the antipattern:

Mixin: Mixin RailwayAsset
Rigid
Subtypes:
    SubKind RailwayLot
    SubKind Terminal

Antipattern Descripton:

This anti-pattern occurs when a «mixin» is specialized only by type whose rigidity meta-property are the same, i.e., only rigid types («kind», «quantity», «collective», «subkind» or «category») or only by anti-rigid types («role», «phase», «roleMixin»)

Would you like to:

use the step-by-step wizard?
[✓ go directly to the refactoring options?]

Figura 4. Anti-pattern management support in Menthor Editor.

6. Final Considerations

In this paper, we extended our work on ontological anti-patterns, proposing three new error-prone structures in combination with pre-defined rectification solutions. In particular, the anti-patterns we describe in this paper regard the formalization of taxonomic structures, the "backbone" of every ontology. Therefore, the identification of these anti-patterns, their associated rectification plans and the model-based computational tool presented here contribute to the theory and practice of ontology engineering.

In companion publications, we presented anti-patterns (with their respective rectification plans) identified in in the modeling of material relations [Sales and Guizzardi 2015], roles [Sales and Guizzardi 2016] and part-whole relations [Sales and Guizzardi 2017]. So, the three anti-patterns related to the modeling of taxonomic structures presented in this paper (MixIden, MixRig and GSRig) come to add to this body of knowledge.In future investigations, we plan to expand this catalog to account for anti-patterns involving other types of entities, in particular phases, qualities and formal relations.

[4]`https://github.com/MenthorTools/menthor-editor`

In terms of frequency of occurrence across the analyzed repository, these three anti-patterns occurred in a frequency much lower than some of the other anti-patterns present in our catalog. For example, MixIden and GSRig appeared in 13,51% of the models in which they could possibly occurr, i.e., in which the necessary modeling elements were present. The MixRig appeared in only one of the models that it could possibly occur (7 in total). However, we found out that in 100% of their occurrences they actually represented modeling errors. This is important because, in these cases, one can actually derive syntactic rules to be encoded in the metamodel of the language such that the occurrence of these anti-patterns would be proscribed in OntoUML models. In other words, the occurrence of these anti-patterns would configure a syntactic error rendering the associated model as a non-valid OntoUML model.

Since anti-patterns signal deviations between intended and valid model instances, and since intended model instances only exist in the mind of domain experts, anti-pattern discovery is a human-centric activity. Hence, the anti-patterns currently making our catalog were discovered in a heavily manual process. To overcome this limitation in our methodology, we intend to study strategies to automate anti-pattern discovery as much as possible. For instance, we would like to provide mechanisms that could automatically learn the recurrent correlation between (a) structures in the unintended model instances, (b) structures in the conceptual models that cause them, and (b) solutions provided by the conceptual modelers over (b) in order to rectify the unintended situation identified in (a). Once these strategies are identified and implemented in our computational support, we intend to extend this tool support to be able to automatically identify these anti-patterns across different conceptual models in our model repository. A possibly promising path for investigation in that respect, in the spirit of [Alrajeh et al. 2015], is the combination of inductive logic learning mechanisms with the counter-example generation capabilities of our model simulation environment (based on Alloy).

Referências

Almeida, J. P. A., Guizzardi, G., and Santos Jr, P. S. (2009). Applying and extending a semantic foundation for role-related concepts in enterprise modelling. *Enterprise Information Systems*, 3(3):253–277.

Alrajeh, D., Kramer, J., Russo, A., and Uchitel, S. (2015). Automated support for diagnosis and repair. *Communications of the ACM*, 58(2):65–72.

Bastos et al. (2011). Building up a model for management information and knowledge: the case-study for a brazilian regulatory agency. In *2nd International Workshop on Software Knowledge (SKY)*, pages 3–11.

Guizzardi, G. (2005). *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology.

Guizzardi, G. (2014). Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In *International Conference on Conceptual Modeling*, pages 13–27. Springer.

Guizzardi, G., Wagner, G., Almeida, J. P. A., and Guizzardi, R. S. (2015). Towards ontological foundations for conceptual modeling: the Unified Foundational Ontology (UFO) story. *Applied ontology*, 10(3-4):259–271.

Jackson, D. (2012). *Software Abstractions: logic, language, and analysis*. MIT press.

Koenig, A. (1995). Patterns and antipatterns. *Journal of Object-Oriented Programming*, 8(1):46–48.

Nardi, J. C., de Almeida Falbo, R., and Almeida, J. P. A. (2013a). Foundational ontologies for semantic integration in eai: a systematic literature review. In *Conference on e-Business, e-Services and e-Society*, pages 238–249. Springer.

Nardi, J. C., Falbo, R. D. A., Almeida, J. P. A., Guizzardi, G., Pires, L. F., van Sinderen, M. J., and Guarino, N. (2013b). Towards a commitment-based reference ontology for services. In *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 175–184. IEEE.

Pereira, D. C. and Almeida, J. P. A. (2014). Representing organizational structures in an enterprise architecture language. In *6th Workshop on formal ontologies meet industry (FOMI)*, volume 1333, pages 7–15. CEUR-WS.org.

Sales, T. P. and Guizzardi, G. (2015). Ontological anti-patterns: empirically uncovered error-prone structures in ontology-driven conceptual models. *Data & Knowledge Engineering*, 99:72–104.

Sales, T. P. and Guizzardi, G. (2016). Anti-patterns in ontology-driven conceptual modeling: the case of role modeling in ontouml. In Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, volume 25, pages 161–187. IOS Press.

Sales, T. P. and Guizzardi, G. (2017). "Is it a fleet or a collection of ships?": Ontological anti-patterns in the modeling of part-whole relations. In *21st European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 28–41. Springer.

Verdonck, M. and Gailly, F. (2016). Insights on the use and application of ontology and conceptual modeling languages in ontology-driven conceptual modeling. In *35th International Conference on Conceptual Modeling (ER)*, pages 83–97. Springer.