Ontological Anti-Patterns in Taxonomic Structures

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

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

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

{tiago.princesales,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 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 this paper are three-fold. Firstly, we propose three 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 software 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 [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., semantic data integration [Padilha et al. 2012]).

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-pattern, namely one characterized by 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 the intended state of affairs in that domain [Guizzardi 2005]. Such a difference occurs either because the model allows some unintended instances or because it forbids some 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 was designed to comply with the ontological distinctions and axiomatization of a theoretically well-grounded foundational ontology named UFO (Unified Foundational Ontology) [Guizzardi 2005]. UFO is an axiomatic formal theory based on theories from formal ontology in philosophy, philosophical logic, 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. The same study also showed that OntoUML is among the most used languages in ontology-driven conceptual modeling [Verdonck and Gailly 2016].

This article complements our earlier work on ontological anti-patterns, in the sense that it puts forth additional error-prone modeling structures. Previously, we focused on anti-patterns that emerged from the modeling of: (i) material relations [Sales and Guizzardi 2015]; (ii) roles [Sales and Guizzardi 2016]; and part-whole relations [Sales and Guizzardi 2017]. Now we focus on anti-patterns that emerge from the modeling of 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 used 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 a model; (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 present the anti-pattern elicitation method and the model benchmark used in this research. In Section 4, we present the newly elicited ontological anti-patterns, including their potential unintended consequences and respective rectification solutions. Section 5 elaborates on the extensions implemented in the OntoUML editor taking into account these anti-patterns. Lastly, Section 6 presents some final considerations.

2. Background: A Brief introduction to UFO and OntoUML

OntoUML is a language whose meta-model was designed to comply with the ontological distinctions and axiomatization of a theoretically well-grounded foundational ontology

named UFO (Unified Foundational Ontology) [Guizzardi 2005]. In the remainder of this section, we briefly explain a subset of UFO's ontological distinctions that are relevant for the anti-patterns discussed in this paper, as well as how these distinctions are reflected in OntoUML's modeling primitives.

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 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, and thus, are deemed *semi-rigid types*. 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 ob-

jects 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

The anti-patterns presented in this paper were identified through an empirical and qualitative method. First, we assembled a repository of ontologies to validate. Then, for each ontology, we selected relevant fragments to inspect. For each fragment, we searched 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; and (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. To decide whether the identified structure was indeed problematic, we would interact with the modelers who created it (if they were available) or inspect the documentation accompanying the model. Upon the confirmation of a modeling 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 would then continue with the simulation. This iteration would be repeated until no more problems could be identified in the fragment at hand. The analysis of a model would stop whenever we had inspected all of its relevant fragments. Lastly, after inspecting each model, we would revisit the identified problem-solution pairs in order to generalize them into anti-patterns and refactoring plans.

We systematically carried out this process for 54 ontologies[2] developed in different types of context: (i) 11 were developed in a pure research environment, such as O3 [Pereira and Almeida 2014], an ontology about organizational structures; (ii) 7 were developed in collaboration with private or public organizations, e.g., the MGIC Ontology [Bastos et al. 2011], a model developed for a Brazilian regulatory agency; (iii) 32 were designed as course assignments in post-graduate ontology engineering courses; and (iv) 4 were developed in other contexts.

These ontologies were created for a variety of purposes: (i) 10 were designed to be reference models of consensus for particular domains (e.g., UFO-S [Nardi et al. 2013] for the service domain); (ii) 10 were developed to assess the appropriateness of existing domain representations, such as database schemas, modeling languages, and standards (e.g., an ontology developed to assess the ITU-T recommendation G. 805 [Barcelos et al. 2011]); (iii) 8 were designed to support the development of knowledge-based applications; (iv) 6 to support semantic interoperability (e.g., OntoBio [Albuquerque et al. 2015], an ontology of the biodiversity domain); (v) 2 for enterprise modeling; (vi) and 26 for other or non-declared purposes.

Moreover, these ontologies were created by modelers with varying levels of ex-

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

[2]For a complete description of the ontologies analyzed in this study, please refer to [Sales and Guizzardi 2015].

pertise 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. Non-Sortal with a Uniform Identity Principle (NSIden)

The anti-pattern named *Non-Sortal with a Uniform Identity Principle (NSIden)* is defined by a non-sortal type that can only be instantiated by individuals that adhere to a single identity principle. Structurally, it corresponds to a type, hereafter *NonSortal*, stereotyped as «mixin», «category» or «roleMixin», whose direct subtypes, hereafter $Sortal_{\text{i}}$, are sortal types that share a common identity provider, hereafter *IdProvider*. This implies that every $Sortal_{\text{i}}$ is stereotyped as «subkind», «role», «phase», «kind», «collective» or «quantity» and that *IdProvider* is stereotyped as a «kind», «collective» or «quantity».

We say that the set of $Sortal_{\text{i}}$ shares an identity provider if: (i) every $Sortal_{\text{i}}$ is stereotyped as «subkind», «role» or «phase» and directly or indirectly specialize *IdProvider*; or (ii) exactly one $Sortal_{\text{i}}$ is stereotyped as «kind», «collective» or «quantity» and the remainder directly or indirectly specialize it. In fact, in this second case, the *IdProvider* is also one of classes playing the role of $Sortal_{\text{i}}$.

To unveil whether a *NSIden* occurrence is indeed a modeling error, one should first assess if *NonSortal* really is a type that can classify instances of different kinds, i.e., that obey multiple identity principles. If that is not the case, the model is wrong and *NonSortal* should actually be represented as a sortal. That can be achieved by (i) replacing the stereotype of *NonSortal* with «subkind», «role» or «phase» and (ii) making *NonSortal* specialize *IdProvider*. This solution is depicted as refactoring plan 1 in Table 1.

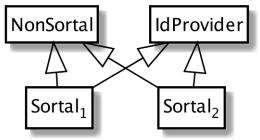
Conversely, if *NonSortal* can indeed classify individuals of different kinds, one should analyze if it is true that *IdProvider* is the kind that supplies the identity principle for every $Sortal_{\text{i}}$. If that is not the case, the model can be rectified by adjusting the kinds accordingly. This second solution is depicted as refactoring plan 2 in Table 1.

If it is the case that *NonSortal* can classify individuals of different kinds and that every $Sortal_{\text{i}}$ inherits their identity principle from the appropriate kind, it means that the model under analysis is incomplete. In other words, *NonSortal* can be instantiated by a type of individual that is not currently accounted for. If ontological completeness is a desired quality for the model at hand, we recommended adding the missing specializations of *NonSortal* as depicted by the refactoring plan 3 in Table 1.

To illustrate the *NSIden* anti-pattern, let us discuss an example found in a reference model developed for the Brazilian ground transportation agency [Bastos et al. 2011]. A fragment of this ontology, related to the domain of highway concessions, is shown in the left side of Figure 1. It depicts two types of organizations, namely *Concessionaire* and *PublicOrganization*. The former represents private companies that are created exclusively to administrate federal highways, whilst the latter accounts for organizations owned and managed by the government, such as regulatory agencies and public foundations. Our main focus here is on the *MaintenanceContractor*, a role played by organizations responsible for (part of) the maintenance of a highway. According to Brazilian law, only public

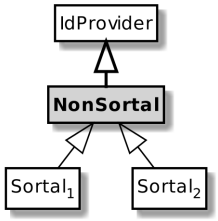
Table 1. Summary of the *NSIden* anti-pattern.

Name (Acronym)		**Description**
Non-Sortal with a Uniform Identity Principle (NSIden)		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	*NonSortal*	«mixin», «category», «roleMixin»
1..*	$Sortal_{\text{i}}$	«subkind», «role», «phase», «kind», «collective», «quantity»
1..*	*IdProvider*	«kind», «collective», «quantity»
Generic Example		

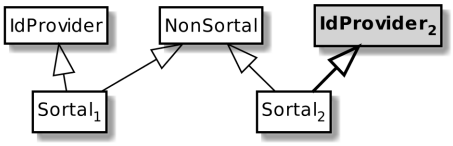


Refactoring Plans

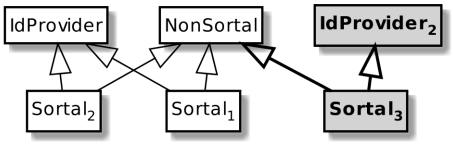
1. From non-sortal to sortal: change the stereotype of *NonSortal* to either «subkind», «role» or «phase» and make it specialize *IdProvider*.



2. Identity adjustment: change the identity provider of at least one $Sortal_{\text{i}}$ to another «kind», «quantity» or «collective» ($IdProvider_2$).



3. Find the missing sortal: add a type, $Sortal_3$, that does not directly or indirectly specialize *IdProvider* as a direct subtype of *NonSortal*.



organizations and concessionaires may have such a responsibility, hence, the two "sub-roles" *PublicContractor* and *ConcessionaireContractor*.

This fragment exemplifies the *NSIden* anti-pattern because *MaintenanceContractor* is represented as a «roleMixin» and all its subtypes, namely *PublicContractor* and

ConcessionaireContractor are of the kind *Organization*. As such, it identifies an ontological mistake because, since only specific types of organizations can be contractors, the *MaintenanceContractor* role is a sortal type. The rectification of this fragment, which follows the *NSIden's* refactoring plan 1, is depicted in the right side of Figure 1. The affected class has a grey background and the new generalization has bolder lines.

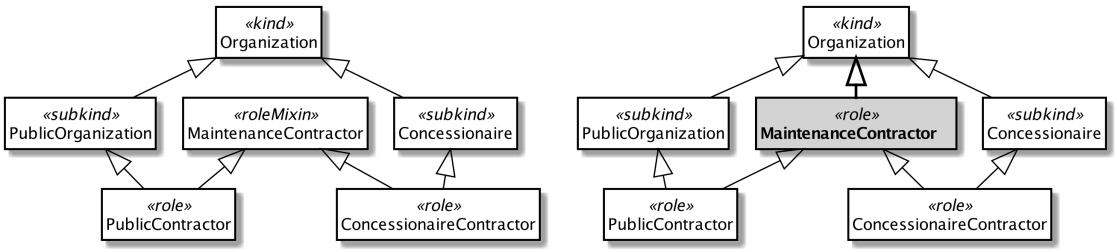


Figure 1. Adapted fragment from the MGIC Ontology exemplifying *NSIden*.

4.2. Mixin with Uniform Rigidity (MixRig)

The *Mixin with Uniform Rigidity (MixRig)* anti-pattern warns modelers about a potential mischaracterization of a type as semi-rigid. It does so by highlighting a «mixin» class, hereafter *Mixin*, whose direct subtypes, hereafter $Type_{\text{i}}$, are either all rigid or all anti-rigid.

To judge whether a *MixRig* occurrence is indeed a modeling mistake, one should start by revisiting the alleged semi-rigid type. Is it actually the case that it characterizes some of its instances essentially and others accidentally? If the answer is no, this type is not semi-rigid, and the model should be rectified according to the refactoring plan 1 described in Table 2. Simply put, if every $Type_{\text{i}}$ is rigid, the rectification consists in changing *Mixin's* stereotype to «category». If every $Type_{\text{i}}$ is anti-rigid, the new stereotype should be «roleMixin».

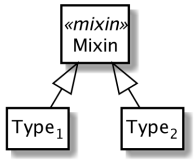
Contrarily, if *Mixin* is indeed a semi-rigid type, the issue may lie on some $Type_{\text{i}}$. One of them may have been modeled using a metaclass that embeds the wrong rigiditiy meta-property. For instance, «subkind» was used instead of «role». In this case, the solution consists in identifying which $Type_{\text{i}}$ has the wrong stereotype and fixing it accordingly. This solution is listed as refactoring plan 2 in. If every $Type_{\text{i}}$ was correctly modeled, it means that the model lacks at least one subtype of *Mixin* with a rigidity property that differs from those of $Type_{\text{i}}$. The solution then, if ontological completeness is desired, would be to add the missing type, as described in the refactoring plan 3.

We illustrate the *MixRig* anti-pattern with another taxonomic fragment extracted from ontology discussed in [Bastos et al. 2011], but now in the sub-domain of railway concession. Depicted in the left-side of Figure 2, it focuses on two types of assets that can be railway system components, namely *Terminal* and *RailwayLot*. A *Terminal* is a type of *Building* where trains stop for passengers to board and disembark, whilst a *RailwayLot* is a delimited piece of land that composes a railway system, such as the areas alongside train tracks.

This example characterizes a *MixRig* anti-pattern because *RailwayAsset* is represented as a «mixin» and its direct subtypes, *Terminal* and *RailwayLot*, are both subkinds, i.e., rigid types. Interacting with the authors of this model, we discovered that it is essential for a *Terminal* to be a part of some railway infrastructure, but accidental for a

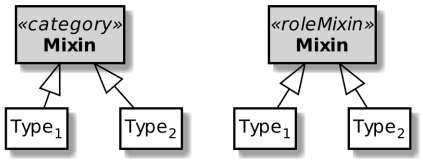
Table 2. Summary of the *MixRig* anti-pattern.

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*$_{\text{i}}$	All class stereotypes except «mixin»
Generic Example		

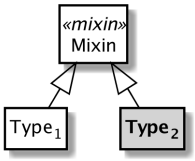


Refactoring Plans

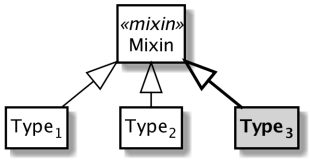
1. Not semi-rigid: change the stereotype of *Mixin* either to a «category», if every *Type*$_{\text{i}}$ is rigid, or to a «roleMixin», if every *Type*$_{\text{i}}$ is anti-rigid.



2. Rigidity adjustment: change the stereotype of at least one *Type*$_{\text{i}}$ (but not all of them) such that the new stereotype embeds a rigidity meta-property different from the remainder *Type*$_{\text{i}}$.



3. Find the missing subtype: specialize *Mixin* with a type ($Type_3$) that has a rigidity meta-property different from every other *Type*$_{\text{i}}$.



Lot. Thus, *RailwayAsset* was properly characterized as a semi-rigid type, but *RailwayLot* was mistakenly represented as «subkind», for it should have been a «role» played by a *Lot* when composing a Railway infrastructure (as described in refactoring plan 2). The rectified fragment is depicted on the right side of Figure 2.

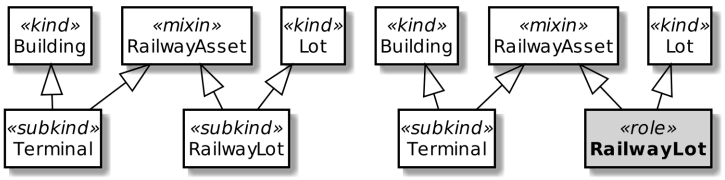


Figure 2. Adapted fragment from the MGIC Ontology exemplifying *MixRig*.

4.3. Generalization Set with a Mixed Rigidity

The *Generalization Set With Mixed Rigidity (GSRig)* anti-pattern aims to identify the use of multiple dividing principles within a single generalization set. Structurally, it consists in a generalization set, hereafter *GS*, that satisfies two conditions: (i) its common supertype, hereafter *RigidSupertype*, is rigid; and (ii) it groups at least one generalization whose subtype is rigid, hereafter $Rigid_{\text{i}}$, and at least one whose subtype is anti-rigid, hereafter $AntiRigid_{\text{i}}$.

As discussed in Section 2, a dividing principle identifies the "dimension" considered for creating a subtype. For instance, the dimension used to differentiate the classes Child, Adult and Elder is the age of a Person. We uncovered throughout our analysis that when multiple dimensions were used to define subtypes within the same generalization set, it was likely that the model misrepresented its domain of interest.

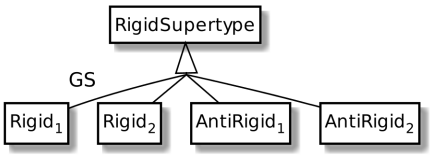
Note that a *GSRig* occurrence will cause a logical contradiction if it contains exactly one $AntiRigid_{\text{i}}$ and the *GS* is set as disjoint and complete. The contradiction arises from the combination of: (i) the rigidity constraints of the subtypes; (ii) the disjointness constraint of the set, 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 "behave" as if it were rigid.

To determine whether a *GSRig* occurrence characterizes a modeling error, one should start by double checking the rigidity of each subtype and fixing it if necessary. If after this step all subtypes are either rigid or anti-rigid, the model has been rectified. Otherwise, one should revisit the common supertype's intended semantics to assess whether it should have been characterized as a «mixin», a metaclass that exactly captures common properties shared by rigid and anti-rigid subtypes. This analysis and rectification alternatives are listed as refactoring plan 1 in Table 3.

In case all types are characterized by the appropriate stereotypes, we recommend one to think about the dividing principle used to create each subtype. If more than one was used, one should revisit the constraints defined by *GS* (e.g., that the subtypes are disjoint). If they were not accurate, the models could commonly be rectified by either: (i) splitting the GS in two, one for the rigid subtypes and another for the anti-rigid ones (refactoring plan 2); or (ii) adding a rigid type as a sibling of the other rigid subtypes and making all

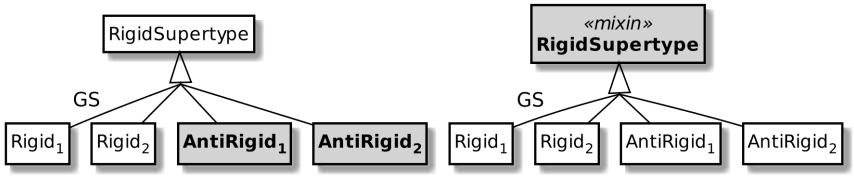
Table 3. Summary of the *GSRig* anti-pattern.

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	*RigidSupertype*	«kind», «collective», «quantity», «subkind», «category»
1..*	$Rigid_i$	«kind», «collective», «quantity», «subkind», «category»
1..*	$AntiRigid_i$	«role», «phase», «roleMixin»
Generic Example		

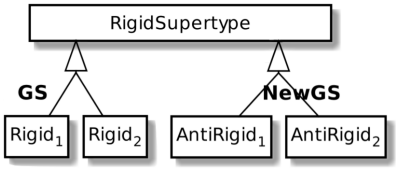


Refactoring Plans

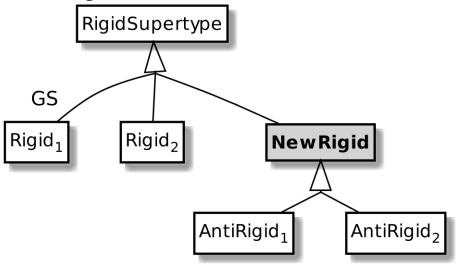
1. Rigidity adjustment: change the stereotype of every *AntiRigid*$_i$ to make them rigid, change the stereotype of every *Rigid*$_i$ to make them anti-rigid, or make *RigidSupertype* a «mixin»



2. Orthogonal generalization set: add a generalization set (GS_2) to group the generalizations of anti-rigid subtypes.



3. Add a rigid subtype: add a rigid subtype (*NewRigid*) that specializes *RigidParent* and is a direct supertype of every *AntiRigid*$_i$ in the generalization set.



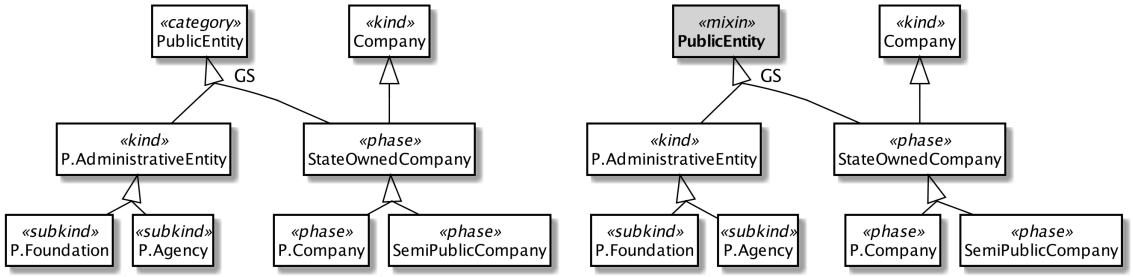


Figure 3. Adapted fragment from the MPOG Ontology exemplifying *GSRig*.

anti-rigid subtypes in the generalization set specialize it (refactoring plan 3).

We exemplify *GSRig* by means of a fragment of a conceptual model about organizational structures published by the Brazilian Ministry of Planning, Budgeting and Management [Ministério do Planejamento, Orçamento e Gestão 2011]. As shown in the left side of Figure 3, this fragment defines the concept of *PublicEntity*, a public organization autonomously managed and with its own independent budget. Such an entity may be of one of two types, namely a *StateOwnedCompany* or a Public Administrative Entity (*P.AdministrativeEntity*). The former refers to companies that are partially or completely owned by the state (e.g., Petrobras, Bank of Brazil), while the latter refers to other public organizations, such as regulatory agencies and public foundations.

The generalization set with *PublicEntity* as its supertype characterizes this *GSRig* occurrence. It aggregates the generalizations coming from *StateOwnedCompany*, an anti-rigid type, and *P.AdministrativeEntity*, a rigid type. From the domain description, we can easily conclude that being a *PublicEntity* is an essential property for some organizations. For instance, it is not possible (and arguably it will never be) for the Ministry of Science and Technology to be privately owned. We can just as easily find examples of organizations that are only contingently public entities, such as a public company that was privatized. Thus, the issue lies in the representation of *PublicEntity* as a category, while in reality, it is a mixin. The rectification of this fragment, following refactoring plan 1, can be seen on the right side of Figure 3.

5. Tool Support

To help modelers validate their ontologies using the anti-patterns presented in this paper, we implemented an anti-pattern management feature in Menthor Editor[3] [Moreira et al. 2016], an open-source ontology-driven conceptual modeling environment. Following the strategy adopted in [Sales and Guizzardi 2015], the anti-pattern feature includes (i) automatic anti-pattern detection; (ii) anti-pattern analysis using a wizard-based strategy; and (iii) automatic model rectification based on the refactoring plans previously presented.

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 partic-

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

ular, the anti-patterns we describe in this paper (NSIden, MixRig and GSRig) regard the formalization of taxonomic structures, the "backbone" of every ontology. Therefore, the identification of these anti-patterns, their associated rectification plans, and the 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 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 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.

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, NSIden and GSRig appeared in 13,51% of the models in which they could possibly occur, i.e., in which the necessary modeling elements were present. MixRig appeared in only one of the models that it could possibly occur (7 in total). However, we found out that 100% of its occurrences were actually 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.

Since anti-patterns signal deviations between intended and valid model instances, and since the former 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 methodological limitation, 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 anti-patterns through the identification of correlations between (a) structures in the unintended model instances, (b) structures in the conceptual models that cause them, and (c) solutions provided by the conceptual modelers over (b) in order to rectify unintended situations identified in (a). In that respect, a possibly promising path for research, in the spirit of [Alrajeh et al. 2015], is to combine inductive logic learning mechanisms with the counter-example generation capability of our Alloy-based model simulation environment.

References

Albuquerque et al. (2015). OntoBio: A biodiversity domain ontology for Amazonian biological collected objects. In *48th Hawaii Int. Conf. on System Sciences*, pages 3770–3779. IEEE.

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

Barcelos et al. (2011). Ontological evaluation of the ITU-T recommendation G. 805. In *18th Int. Conf. on Telecommunications*, pages 232–237. IEEE.

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*, 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 et al. (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.

Ministério do Planejamento, Orçamento e Gestão (2011). Esboço de modelagem conceitual para estruturas organizacionais governamentais brasileiras e o SIORG.

Moreira et al. (2016). Menthor editor: An ontology-driven conceptual modeling platform. In *2nd Joint Ontology Workshops*, volume 1660. CEUR-WS.org.

Nardi et al. (2013). Towards a commitment-based reference ontology for services. In *17th Int. Enterprise Distributed Object Computing Conference*, pages 175–184. IEEE.

Padilha et al. (2012). Ontology alignment for semantic data integration through foundational ontologies. In *31st Int. Conf. on Conceptual Modeling*, pages 172–181. Springer.

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*, 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.