Ontological Anti-Patterns: Empirically Uncovered Error-Prone Structures in Ontology-Driven Conceptual Models

Tiago Prince Sales, Giancarlo Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO),
Computer Science Department, Federal University of Espírito Santo (UFES), Vitória - ES, Brazil
tpsales@menthor.net, gguizzardi@inf.ufes.br

Abstract. The construction of large-scale reference conceptual models is a complex engineering activity. To develop high-quality models, a modeler must have the support of expressive engineering tools such as theoretically well-founded modeling languages and methodologies, patterns and anti-patterns and automated supporting environments. This paper proposes a set of Ontological Anti-Patterns for Ontology-Driven Conceptual Modeling. These anti-patterns capture error-prone modeling decisions that can result in the creation of models that fail to exclude unintended model instances (representing unintended state of affairs) or forbid intended ones (representing intended states of affairs). The anti-patterns presented here have been empirically elicited through an approach of conceptual models validation via visual simulation. The paper also presents a series of refactoring plans for rectifying the models in which these anti-patterns occur. In addition, we present here a computational tool that is able to: automatically identify these anti-patterns in user's models, guide users in assessing their consequences, and generate corrections to these models by the automatic inclusion of OCL constraints implementing the proposed refactoring plans. Finally, the paper also presents an empirical study for assessing the harmfulness of each of the uncovered anti-patterns (i.e., the likelihood that its occurrence in a model entails unintended consequences) as well as the effectiveness of the proposed refactoring plans.

Keywords: Ontology-Driven Conceptual Modeling, Ontological Anti-Patterns, OntoUML, UFO

1. Introduction

Information is the foundation of all rational decision-making. Without suitable *Information Systems*, individuals, organizations, communities and governments can neither systematically take optimal decisions nor understand the full effect of their actions. We use the term *Information System* here in a broader sense that includes also *Socio-technical Systems*. Moreover, we subscribe here to the so-called *Representation View* of information systems [1]. Following this view, an information system is a representation of a certain *conceptualization* of reality. To be more precise, an information system contains information structures that represent *abstractions* over certain portions of reality, capturing aspects that are relevant for a class of problems at hand. In this view, the quality of an information system directly depends on how truthful are its information structures to the aspects of reality it purports to represent.

In his ACM Turing Award Lecture entitled *"The Humble Programmer"* [2], E. W. Dijkstra discusses the sheer complexity one has to deal with when programming large computer systems. His article represented an open call for an acknowledgement of the complexity at hand and for the need of more sophisticated techniques to master this complexity. Dijkstra's advice is timely and even more insightful in our current scenario, in which semantic interoperability becomes a pervasive force driving and constraining the process of creating information systems in increasingly complex combinations of domains. More and more, information systems are created either by combining existing independently developed subsystem, or are created to eventually serve as components in multiple larger yet-to-be-conceived systems. In this scenario, information systems engineering, in particular, and rational governance, in general, cannot succeed without the support of a particular type of discipline. A discipline devoted to establishing well-founded theories, principles, as well as methodological and computational tools for supporting us in the tasks of understanding, elaborating and precisely representing the nature of conceptualizations of reality, as well as in tasks of negotiating and safely establishing the correct relations between different conceptualizations of reality. On one hand, this discipline should help us in producing representations of these conceptualizations that are *ontologically consistent*, i.e., that represent a worldview that aggregates a number of abstractions that are consistent with each other. On the other hand, it should help us to make explicit our *ontological commitments*, i.e., to make explicit what exactly is the worldview to which we are committing. In summary, this discipline should help to produce concrete representation artifacts (models) of conceptualizations of reality that achieve the goals of *intra-worldview consistency* and *inter-worldview interoperability*.

The discipline to address the aforementioned challenges is the discipline of *Conceptual Modeling*. However, in order to do that, conceptual modeling languages, methodologies and tools must be informed by another discipline, namely, the discipline of *Ontology*, in philosophy. *Formal Ontology* has exactly the objective of developing domain-independent theories and systems of categories and their ties that could then be used to articulate conceptualizations in different domains in reality. More recently, the discipline of *Applied Ontology* has developed systematic and repeatable techniques for applying these theories in solving problems in concrete domains [3]. Given this essential role played by Ontology in this view of the discipline of Conceptual Modeling, we term it *Ontology-Driven Conceptual Modeling*.

The importance of Ontology as a foundation for Conceptual Modeling is not new in this discipline. There is an established tradition and a growing interest in using ontological theories for analyzing conceptual modeling languages as well as for proposing methodological guidelines for using these languages in the production of ontologically consistent models [1,4-6]. Ontology has been used not only as an analysis tool but also in the development of engineering tools such as conceptual modeling languages with explicitly defined and properly axiomatized metamodels [7], as well as computational environments supporting automated model verification, validation and transformation [8,9]. These are complexity management tools that are fundamental for addressing the challenge highlighted by Dijkstra's advice.

In the invited paper [10] companion to his keynote talk delivered at the 2014 edition of the International Conference on Conceptual Modeling (ER), the second author of the present paper makes the case for a particular set of complexity management tools needed for Ontology-Driven Conceptual Modeling, namely *Ontological Conceptual Patterns*, *Ontological Anti-Patterns*, and *Ontology Pattern Languages*. In the present paper, we focus on one of these conceptual tools, namely, *Ontological Anti-Patterns*.

An anti-pattern is a recurrent error-prone modeling decision [11]. 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 [12]. We name these configurations *Ontological Anti-Patterns*.

In this paper, we focus on the study of Ontological Anti-Patterns in a particular conceptual modeling language named OntoUML [7]. OntoUML is an example of a conceptual modeling 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) [13]. UFO is an axiomatic formal theory based on theories from Formal Ontology in Philosophy, Philosophical Logics, Cognitive Pyschology and Linguistics. OntoUML has been successfully employed in a number of industrial projects in several different domains, such as petroleum and gas, digital journalism, complex digital media management, off-shore software engineering, telecommunications, retail product recommendation, and government. Besides the modeling language itself, the OntoUML approach also offers a model-based environment for model construction, verbalization, code generation, formal verification and validation [8,9]. In particular, the validation strategy employed there makes use of an approach based on visual model simulation [9]. In this paper, we make use of this approach for eliciting *Ontological Anti-Patterns* in OntoUML.

This paper can also be seen as an extension of another publication presented at the same edition of the ER conference in which [10] was presented, namely [14]. In comparison to [10], this paper is much narrower in scope but much deeper in its investigation regarding ontological anti-patterns; in comparison to [14], we have expanded the paper in the following manner. Firstly, we have used an enlarged model repository with two added conceptual models. We here also present a much more detailed characterization of the repository and a more detailed analysis of an empirical study for uncovering the Ontological Anti-Patterns. Secondly, we present here a very detailed definition for the set of Anti-Patterns uncovered by this study, precisely defining their characteristics and structures and providing examples of their occurrences in the models in the repository. More importantly, we define here formal refactoring plans that have been implemented in a computational tool for automatically rectifying the potential modeling mistakes correlated with the presence of these anti-patterns. In this paper, we also present a novel industrial empirical study, which analyzes in depth the largest conceptual modeling in our repository with the goal of establishing: (i) the likelihood that an anti-pattern entails unintended consequences in the model; (ii) the effectiveness of our proposed refactoring plans. The paper also presents a newer implementation of the tool (in comparison to the one presented in [10] and [14]) that incorporates a model wizard for helping the modelers in applying the proposed refactoring plans.

The contributions of this paper are three-fold. Firstly, we contribute to the identification of Ontological Anti-Patterns in Conceptual Modeling. We do that by carrying out an empirical qualitative approach over a model benchmark of 54 OntoUML models. In particular, we employ the visual simulation capabilities embedded in OntoUML editor [8]. Secondly, after precisely characterizing these anti-patterns, we propose a set of refactoring plans that can be applied to the models in order to eliminate the possible unintended consequences induced by the presence of each of these anti-patterns. Finally, we present an extension to the OntoUML editor with a number of features for: (a) automatically detecting anti-patterns in user models; (b) supporting the user in exploring the consequences of the presence of an anti-pattern in the model and, hence, deciding whether that anti-pattern indeed characterizes a modeling error, either because it allows unintended model instances or because it forbids intended ones; (c) automatically executing refactoring plans to exclude these unintended model instances, which can take the form of OCL constraints or direct interventions in the model.

The remainder of this paper is organized as follows: in Section 2, we briefly elaborate on the modeling language OntoUML and some of its underlying ontological categories, as well on the approach for model validation via visual simulation embedded in the OntoUML editor; Section 3 characterizes the model benchmark used in this research; Section 4 presents the elicited Ontological Anti-Patterns with their unintended consequences as well possible solutions for rectification in terms of model refactoring plans; section 5 elaborates on the extensions implemented in the OntoUML editor taking into account these anti-patterns. Section 6 presents an additional industrial empirical study aimed at investigating the accuracy of our anti-pattern catalog as well as the efficacy of the proposed refactoring plans. Finally, section 7 presents some final considerations including a brief discussion on related work.

2. Model Validation via Visual Simulation in OntoUML

The OntoUML language meta-model contains: (C1) elements that represent ontological distinctions prescribed by the underlying foundational ontology UFO; (C2) constraints that govern the possible relations that can be established between these elements reflecting the axiomatization of this underlying foundational ontology UFO. These two characteristics are illustrated below using some ontological distinctions among the categories of object types (e.g., *Kind, Subkind, Roles and RoleMixins*), trope types (e.g., *Relator*, *Mode*) and relations (*formal relations, material relations* and *parthood relations*). For an in depth presentation, formal characterization and empirical evidence for a number of the ontological categories underlying OntoUML, the reader is referred to [7].

In UFO's theory of types, we have a fundamental distinction between what are named *Sortal* and *Non-Sortal types*. A sortal is a type whose instances obey a uniform principle of identity. A principle of identity, in turn, is a principle with which we can judge if two individuals are the same or, as a special case, what changes an individual can undergo and still be the same. A stereotypical example is the type Person. Contrast it with the type Insurable Item. Whilst in the former case all instance of that type obey the same principle of identity, in the latter case, the type classifies instances of different kinds (e.g., cars, boats, people, houses, body parts, works of art) and that obey different principles of identity. A *Kind* is a sortal that is rigid and that *supplies* a uniform principle of identity for all its instances. As formally demonstrated in [7], every object in a conceptual model must obey a unique principle of identity and, hence, must be an instance of a unique kind. *Subkinds* are rigid specializations of a Kind and inherit that principle of identity supplied by that unique subsuming Kind. Rigidity can be characterized as follows: a type T is rigid iff all instances of that type are necessarily (in the modal sense) instances of that type, i.e., the instances of T cannot cease to be an instance of T without ceasing to exist.

In contrast with rigidity, we have the notion of anti-rigidity: a type T' is anti-rigid iff every instance of that type can cease to be an instance of that type (again, in the modal sense), i.e., instances of T' can move in an out of the extension[1] of T' in different possible worlds while maintaining their identity. Among the anti-rigid sortal types, we have the subcategory of *Roles*. Besides being anti-rigid, the Role category possesses another meta-property named *Relational Dependence* [7]. According to this meta-property, instances of roles move in and out of the extension of that type due to a change in one of its *relational properties*, i.e., due to the establishment or termination of a *relation*. For instance, a student is a role that a person plays *when related to* an education institution, and it is the establishment (or termination) of this relation that alters the instantiation relation between an instance of person and the type Student. Analogously, a husband is a role played by a person *when married to* a (person playing the role of) wife.

Distinctions generated by the variation of these ontological meta-properties (e.g., rigidity, relational dependence) can also be found among non-sortals. One example is the notion of a *RoleMixin*. A RoleMixin is a non-sortal, which is also anti-rigid and relationally dependent. In other words, the RoleMixin category is similar to and, hence, is subject to many of the same constraints of the Role category. However, unlike a role, a RoleMixin classify entities that instantiate different kinds (and that obey different principles of identity). A classical example

[1]The extension of a type t in a world w, logically represented by the expression $ext_w(t)$, stands for the set of all individuals that instantiate the type t in the world w.

of a RoleMixin is the type Customer that can be instantiated by instances of the kinds Person and Organization [7]. In this example, Customer is anti-rigid (i.e., no Customer is necessarily a Customer), relationally dependent (i.e., in order to be a Customer someone has to purchase something from a Supplier) and entities of different kinds can be Customers, namely, persons and organizations.

In UFO, there is also fundamental distinction between the so-called *formal* and *material relations*. A formal relation is a relation that holds directly between its relata and that is reducible to intrinsic properties of these relata. Take, for instance, the relation of being-taller-than between people. If John is taller than Paul then this relation is established by the mere existence of John and Paul. Moreover, in this case, there is no real connection between John and Paul, but the relation is reducible to intrinsic properties of these two individuals, namely, John is taller than Paul iff John's height is bigger than Paul's height. Now, take the case of relations such as being-married-to, being-enrolled-at, being-employed-by, being-a-customer-of, etc. These relations are not reducible to intrinsic properties of their relata. In contrast, in order for these relations to hold, something else needs to exist connecting their relata, namely, particular instances of marriages, enrollments, employments and purchases. These mediating entities can be thought as aggregations of relational properties and are termed *relators* [7]. Relations that are founded on these relators are termed *material relations*.

Relators are examples of existentially dependent entities. In fact, they are entities that are existentially dependent on a multitude of individuals. For instance, marriages, contracts, employments and covalent bonds require the existence of multiple entities in order to exist. UFO also countenances the existence of entities that are existentially dependent on single individuals. Suppose, for instance, that John has a case of Dengue Fever. This entity (John's Dengue Fever) is a complex entity can have its own properties and that can change in time maintaining its identity. However, it is an ontologically dependent entity: it existentially depends on John. In UFO, these entities are named *modes*. Modes are connected to the their bearers (i.e., the entities they dependent on) by a formal relation named *characterization* (a type of existential dependence relation) in an analogous manner in which relators are connected to their bearers by a formal relation named *mediation* [7].

Finally, regarding parthood relations, UFO has a rich system of parthood relations [7]. Firstly, this system differentiates parthood relations in terms of the type of relata they accept. These include relations between: (i) *quantities* (e.g., portions of wine, water, sand), namely, the so-called *subQuantityOf* relation; (ii) relations involving *collectives* (e.g., a collection of books, a pile of bricks, a forest) namely, the *memberOf* relation and the *subCollectiveOf* relation; (iii) as well as parthood relations in which parts of different types contribute in different ways to the functional behavior of the whole, termed *componentOf* relation. The lattice of parthood theories in UFO also offers a formal characterization for these constructs in terms of their basic meta-properties (e.g., whether they are transitive or not and in which context) as well as in terms of their modal meta-properties (e.g., whether they entail *existential dependence* from the whole to the part - termed *essential parts*, whether they entail existential properties from the part to the whole - termed *inseparable parts*, or both [7]).

In summary, with respect to characteristic (C1) in the beginning of this section, OntoUML incorporates modeling constructs that represent all the aforementioned ontological categories (among many others) as modeling primitives of the language. Regarding (C2), the meta-model embeds constraints that govern the possible relations to be established between these categories. These constraints are derived from the very axiomatization of these categories in the foundational ontology UFO. Examples include (among many others) [7]: a sortal is either a Kind or it must be a subtype of exactly one ultimate Kind; an anti-rigid type cannot be a supertype of a rigid type; a sortal type cannot be a supertype of a non-sortal type, among many others. In fact, as demonstrated in [10], given their associated formal constraints, the OntoUML constructs representing ontological distinctions can only appear in a model forming particular predefined configurations. These configurations are termed in [10] *Ontological Design Patterns*. An example of such a pattern is the *Relator-Material Relation Design Pattern* briefly explained as follows.

In OntoUML, a material relation appears in a model connected to a relator from which it is derived forming the pattern depicted in **Figure 1**. In this pattern, the dashed relation is termed *derivation* and connects a material relation with the relator from which it is derived; the *mediation* relation is a relation of existential dependence connecting an instance of a relator with multiple entities of which a relator depends (e.g., the marriage between Paul and Mary existentially depends on Paul and Mary; the employment between John and the UN likewise can only exist whilst John and the UN exist). Moreover, the cardinality constraints of the derived material relation and of the derivation relation are constrained by the cardinality constraints of these (otherwise implicit) mediation relations (some of these constraints are illustrated in **Figure 1**) [7]. As discussed in [7], the explicit representation of relators solves a number of conceptual modeling problems, including the classical problem of the *collapse of cardinality constraints*. Furthermore, as demonstrated in [16], relators also play a decisive role in providing precise methodological guidelines for systematically choosing between the constructs of *association specialization*, *subsetting* and *redefinition*.

Because of the constraints embedded in the OntoUML metamodel, as discussed in [7], the only grammatically correct models in that language are ontologically consistent models. In other words, by incorporating ontological constraints in its meta-model, OntoUML proscribes the representation of ontologically non-admissible states of affairs in conceptual models represented in that language. However, as discussed in [14], the language cannot guarantee that, in a particular model, only model instances representing *intended state of affairs* are admitted. This is because the admissibility of domain-specific states of affairs is a matter of factual knowledge, not a matter of consistent possibility [10]. In other words, a domain independent language system of representation cannot rule out unintended model instances that are unintended not due to ontological constraints but to domain-specific rules.

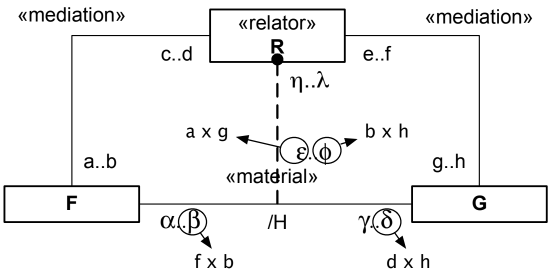


Figure 1. The Relator-Material Relations Pattern (from [7]).

To illustrate the latter point, we will use for the remainder of the paper the running example presented in **Figure 2**. This model describes people's roles and relevant properties in the context of a Criminal Investigation. Some of roles may be the Detectives that investigate the crime, other the Suspects of committing the crime, but also Witnesses that are interrogated by the Detectives about the crime. Each Investigation has a Detective who is responsible for it (referred to as Lead Detective). Detectives are ranked as Junior or Senior Detectives, according to their experience in the job. Since other relational properties are relevant in investigations, the model also represents friendship ("person isFriendOf person"). This model also contains an example of a parthood relation between a Criminal Investigation, the whole, and an Interrogation, the part. Finally, it contains an instance of a variation of the pattern of **Figure 1** involving the Criminal Investigation (relator) and the roles of Detective, Lead Detective, Witness and Suspect as types of relata. The material relations generated from the relators Interrogation and Criminal Investigation and their corresponding mediation relations are omitted here.

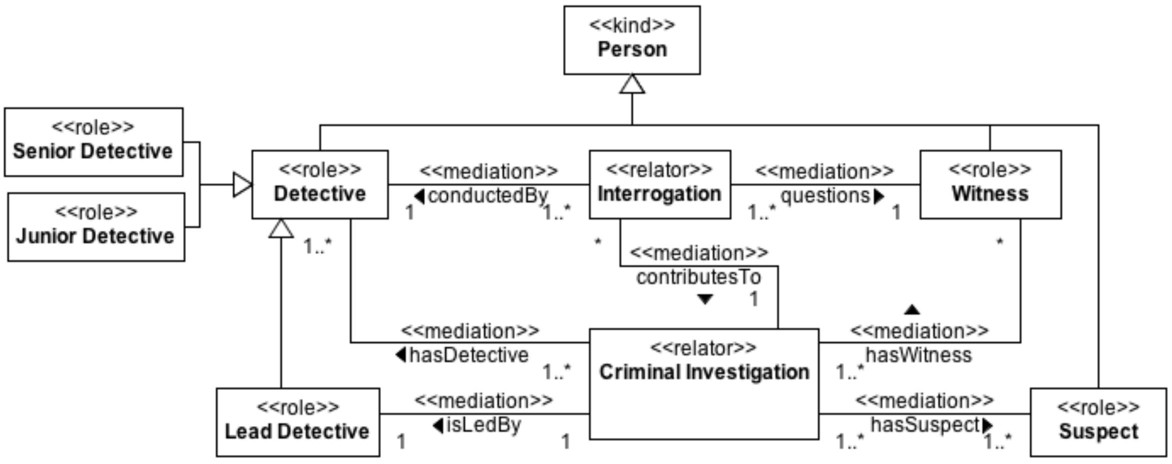

<<kind>> Person
<<role>> Senior Detective
<<role>> Junior Detective
<<role>> Detective
<<relator>> Interrogation
<<role>> Witness
<<mediation>> ◀conductedBy 1 1..*
<<mediation>> questions▶ 1..* 1
<<mediation>> contributesTo ▼ 1
1..*
*
*
<<mediation>> ◀hasDetective 1..*
<<relator>> Criminal Investigation
<<mediation>> ▲ hasWitness 1..*
<<role>> Lead Detective
<<mediation>> ◀isLedBy 1 1
<<mediation>> hasSuspect▶ 1..* 1..*
<<role>> Suspect

Figure 2. Partial OntoUML model of the domain of criminal investigation.

The model of **Figure 2** does not violate any ontological rules in capturing aspects of the investigation domain. It would have done so, for example, had we placed Suspect as a supertype of Person, or had we represented the possibility of a Suspect or Witness without being related to Criminal Investigation (we assume here a Suspect is a suspect in the context of an Investigation and so is a Witness) [7]. These cases can be easily detected and proscribed by an editor such as the one proposed in [8]. Nonetheless, this model admits *many grammatically valid instances that do not represent intended state of affairs*. One example is one in which the Lead Detective of an investigation is also a Suspect on that investigation. Another example is one in which a Detective interrogates himself. A third one is one in which a detective conducts an interrogation in an investigation in which this Detective does not participate. This simple and relatively small model fragment actually contains 9 cases of what we term *Ontological Anti-Patterns*, i.e., model fragments that when used, typically create a deviation between the set of possible (valid) and the set of intended state of affairs [12]. We will return to this point in Sections 4 and 5.

Guaranteeing the exclusion of unintended states of affairs without a computational support is a practically impossible task for any relevant domain [9,10]. In particular, given that many fundamental ontological distinctions are modal in nature, in order to validate a model, one would have to take into consideration the possible valid instances of that model in all possible worlds.

In [9], the authors propose an automated approach for OntoUML that offers a contribution to this problem by supporting conceptual model validation via visual simulation. In the proposed tool, the models are translated into Alloy [15], a logic language based on set theory, which is supported by an analyzer that, given a finite context, exhaustively generates possible instances for a given specification and also allows automatic checking of assertions' consistency. The generated instances of a given conceptual model are organized in a branching-time temporal structure, thus, serving as a visual simulator for the possible dynamics of entity creation, classification, association and termination. In [9], the modeler is then confronted with a visual representation of the snapshots in this world structure. These snapshots represent model instances that are deemed admissible by the ontology's current axiomatization. This enables modelers to detect unintended model instances (i.e., model instances that do not represent intended state of affairs) so that they can take the proper measures to rectify the model. In order to illustrate this point, in figure 3 below we show a model instance for the OntoUML conceptual model of **Figure 2**.

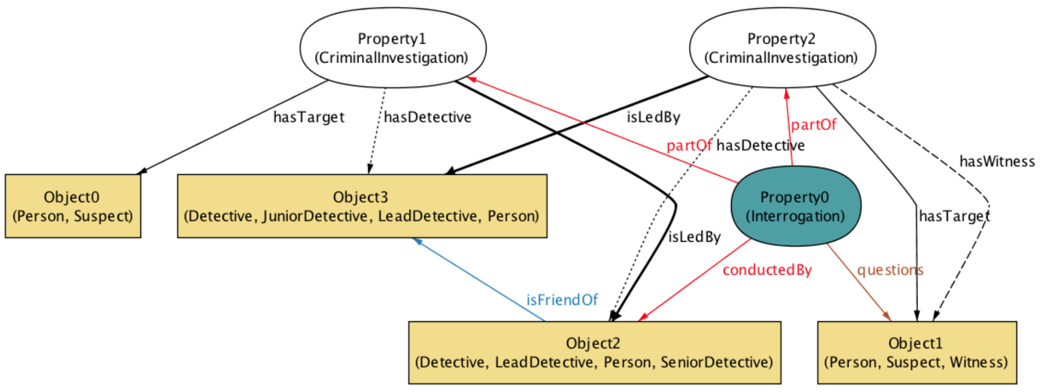


Figure 3. Possible instantiation of the Criminal Investigation model depicted in **Figure 2**.

The comparison between possible (valid) model instances, generated by the Alloy Analyzer, and the intended ones, obtained from domain experts or the conceptual model documentation, highlights possibly erroneous modeling decisions. The recording and categorization of these decisions for a set of OntoUML conceptual models served as a basis for identifying the ontological anti-patterns discussed in this paper. The process for empirically uncovering these anti-patterns is explained in the following, throughout Section 3.

3. Empirically Uncovering Ontological Anti-Patterns

The approach used in this work for the identification of the proposed set of anti-patterns was an empirical qualitative analysis. The idea was to simulate existing OntoUML conceptual models by employing the approach described in Section 2. In a preliminary analysis reported in [17], we studied the recurrence of these anti-patterns across: (i) different domains; (ii) different levels of modeling expertise in Ontology-Driven Conceptual Modeling; (iii) models of different sizes, maturity and complexity.

In our preliminary study, we selected nine models: (1) a conceptual model about the Brazilian health organization system; (2) a conceptual model of the organizational structure of Brazilian federal universities; (3) a conceptual model that describes a domain of online mentoring activities; (4) a domain ontology representing the domain of transport optical network architectures [18]; (5) an ontology in the biodiversity domain [19]; (6) a heart electrophysiology reference ontology [20]; (7) an ontology in the domain of normative acts; (8) an ontology of public tenders; and (9) an ontology in the domain of Brazilian federal organizational structures [21]. Although all models were designed in OntoUML, we distinguish of them between domain ontologies and conceptual models, to distinguish between representations that are actually meant to capture a shared domain conceptualizations, and formalizations of the perception of a particular individual.

Our initial study also took into account the modeler's expertise in using OntoUML and its foundations. We classified them as "beginners" if they had less than one year of experience, and as "experienced" if they had more. Notice, however, that for all the selected models, the individuals involved in their creation had significant experience in traditional conceptual modeling approaches (e.g., UML and ER). In our first sampling of models, we had 4 models created by beginners (models 1-3, 9) and 5 models created by experienced modelers (4-8).

We also classified the investigated models regarding the context of their creation. Three of them were graduate final assignments (models 1-3), two of which were produced by modelers with vast experience in the respective domains (1-2). Model 4 was produced by experienced modelers in an industrial project, who had access to domain experts as well as a supporting international standard of the domain (ITU-T G.805). In fact, the resulting ontology was published in a relevant scientific forum in the area of Telecommunications [18]. Model 5 was developed in the Brazilian National Center for Amazon Research in collaboration with domain experts to support interoperability of biodiversity data [19]. Model 6 was published in a renowned international journal in the area of Bioinformatics in a special issue of Biomedical ontologies [20]. Models 7 and 8 were produced in a large-scale industrial project for the Brazilian Regulatory Agency for Land Transportation (ANTT) [22], in which the modelers had constant access to normative documentation and to domain experts. Lastly, model 9 was produced by a group of modelers in the Brazilian Ministry of Planning (MPOG) [21] that was formed by experts in the domain who had a professional-level experience in traditional conceptual modeling.

Lastly, we registered the size of the models in our initial set. The modelers classified as beginners (models 1-3 and 9) produced models varying from 15-31 classes and 7-30 associations, whilst the experienced modelers designed models containing 46-194 classes and 29-122 associations.

In what follows, we describe our strategy for identifying anti-patterns across our initial sample. For each model, we started by simulating it using the approach described in the previous section. This process resulted in a number of *possible instances* for that model, automatically generated by the Alloy Analyzer. We then contrasted the set of possible instances with the set of *intended instances* of the model, i.e., the set of model instances that represented intended state of affairs according the creators of the models. Whenever we detected a mismatch between these two sets (either the existence of a possible instance that was not intended or the absence of an intended instance), we registered an error in the model. For each registered error, we analyzed the model in order to identify which constructs (or combination thereof) were responsible for causing the error at hand. After simulating all nine models and verifying all errors, we identified and catalogued as anti-patterns those model structures that recurrently produced such mismatches, i.e., modeling patterns that would repeatedly produce model instances that were not intended (i.e., *underconstrained models*) or would exclude instances that were intended (i.e., *overconstrained models*). To be more precise, we considered as anti-patterns those error-prone modeling decisions that occurred in at least one third of the validated models. We carried out this simulation-based validation process with a constant interaction with the model creators (when available) and/or by inspecting the textual documentation accompanying the models.

In this first empirical study, we manage to identify six initial ontological anti-patterns. Two of them were identified in three models. Another three were identified in six, seven and eight models respectively, and one anti-pattern that appeared in all the analyzed models. For more details, please refer to the preliminary report [17]. This initial study gave us confidence that the adopted method could be used as a means for detecting these ontological anti-patterns. In the follow-up study reported here (which extends our initial report in [14]), we manage to assemble a much larger benchmark of 54 OntoUML models. **Table 1** provides a general description of all conceptual models in the repository, following the same classification adopted in the preliminary study[2].

Regarding the development context, most models in this extended sample are graduate course assignments, a total of 32 or 59% of the repository to be more precise. These models were produced as final assignments in a 60-hours OntoUML graduate course. Moreover, 11 models (20%) were developed in academic research without industry collaboration. An example is the *Configuration Management Task Ontology* (CMTO) [23]. Seven models had a participation of private companies and/or governmental organizations. Amongst them, there is the MGIC Ontology [22], the most significant model in our repository. It was developed in the context of a research project with the Brazilian Agency for Ground Transportation (ANTT), which is responsible for regulating ground transportation in Brazil. For the remainder models, we had no knowledge of the model development context.

Concerning the purpose for which the models were created, the repository contains 10 models (16%) intended to serve as reference models of their respective domains. An example is the Core Ontology of Services, named UFO-S [24]. Another 10 models (16%) were developed in order to perform ontological analysis on existing domain formalizations, databases or modeling languages. An example is the refactoring of the Conceptual Schema of Human Genome [25]. Moreover, we had 8 models (13%) designed for the development of knowledge-based

[2]The conceptual models in this table are publicly available at: http://www.menthor.net/model-repository.html. The few exceptions of missing models are due to non-disclosure agreements that prohibited their publication.

applications, 6 models (10%) whose main intention was to support semantic interoperability between systems and/or organizations, and 2 models (3%) intended for enterprise modeling. For the remainder 26 models (42%), we could not retrieve their motivation either from the accompanying text or from their authors.

Concerning the modeler's overall expertise on OntoUML and Ontology-driven Conceptual Modeling, we identify 22 models (41%) developed by beginners (from which 18 are also graduate course assignments) and 32 (59%) developed by experienced modelers. Finally, we look into the number of modelers involved in the ontology design. A single person participated in the development of most of the cases (35 models, roughly 65%). Moreover, 15 models are the product of collaboration efforts between 2-4 people, whilst 4 models involved 7-10 modelers. Of the 35 models developed by a single modeler, 31 were graduate course assignments.

Table 1. Summary description of all models in the repository (subtitle: "Exp." stands for Level of Expertise, whose values are adv = advanced and beg = beginner; "#Md" stands for number of modelers that participated in the development of the model).

Model	Domain	Context	Purpose	Exp.	#Md
The MGIC Ontology [22]	Brazilian Ground Transportation Regulation	Government Project	interoperability; enterprise model.	adv	10
The G.805 Ontology [18]	ITU-T G.805 Recommendation	Industry Project	ontol. analysis; reference model	adv	4
The G.805 Ontology 2.0	ITU-T G.805 Recommendation	Industry Project	reference model; kb application	adv	3
The G.800 Ontology	ITU-T G.800 Recommendation	Industry Project	reference model; kb application	adv	3
OntoEmerge [26, 27]	Emergency Plans	MSc Dissertation	kb application	adv	8
OntoUML Org Ontology (O3) [28, 29]	Enterprise architecure	MSc Dissertation	ontol. analysis; enterprise model	adv	2
The ECG Ontology [20]	Eletrocardiogram	MSc Dissertation	interoperability; kb application	adv	3
Gi2MO Ontology Refactored	Generic idea and innovation management	Graduate Course Assignment	ontol. analysis	adv	1
Internal Affairs Ontology Refactored	Brazilian police internal affairs dept.	Graduate Course Assignment	ontol. analysis	adv	3
Open proVenance Ontology [30]	Provenance of scientific experiments	PhD Thesis	reference model	adv	3
The Library Model	Library archive and services	Graduate Course Assignment	unspecified	beg	1
OntoBio [19]	Amazonian biodiversity	Government Project	interoperability; kb application	adv	3
The Public Tenders Model	Brazilian public tenders	Graduate Course Assignment	unspecified	adv	1
UFO-S [24]	Commitment-based Service	PhD Thesis	reference model	adv	7
The TM Forum Model	Network management	Other	unspecified	beg	1
The Social Contract Model	Brazilian social contract theory	Graduate Course Assignment	unspecified	beg	1
The Clergy Model	Catholic clergy	Graduate Course Assignment	unspecified	beg	1
The FIFA Football Model	Football (based on FIFA's offical rules)	Graduate Course Assignment	unspecified	adv	1
The PAS 77:2006 Ontology [31]	Service continuity	MSc Dissertation	ontol. analysis; reference	adv	4
IDAF Model	Institute of Agriculture Protection of Espírito Santo	Graduate Course Assignment	unspecified	adv	1
The Cloud Vunerability Ontology [32]	IaaS perspective on public cloud vulnerability	MSc Dissertation	reference model	adv	4
The University Model	Brazilian federal universities	Graduate Course Assignment	unspecified	beg	1
Configuration Management Task Ontology (CMTO) [23]	Configuration management of software products	MSc Dissertation	interoperability	adv	2
GRU MPS.BR Model	Reuse management process of MPS.BR	Graduate Course Assignment	unspecified	beg	1
The Experiment Model	Scientific experiment	Graduate Course Assignment	unspecified	beg	1
CSHG Refactored [25]	Human genome	PhD Thesis	ontol. analysis	adv	3
The Normative Acts Ontology [33]	Brazilian normative acts composition	Government Project	reference	adv	3

The Parking Lot System	World view of a parking lot management system	Graduate Course Assignment	unspecified	adv	1
The School Transportation Model	World view of a system to support student transportation	Other	application	beg	1
The Quality Assurance Ontology	Quality assurance based on CMMI, MPS.BR and ISO 9001	Graduate Course Assignment	unspecified	adv	1
The OpenFlow Ontology [34]	OpenFlow communication protocol	BSc Monograph	kb application	beg	2
The Music Ontology Refactored3	Music-related data	Graduate Course Assignment	ontol. analysis	adv	1
The Internship Model	Legal brazilian intership	Graduate Course Assignment	unspecified	adv	1
The G.809 Model	ITU-T G.809 Recommendation	Graduate Course Assignment	unspecified	beg	1
The ERP System Model	World view of an enterprise resource planner system	Other	interoperability	adv	1
The Online Mentoring Model	World view of a system to support online mentoring	Graduate Course Assignment	unspecified	adv	1
The Help Desk System Model	A model that describes a potential help desk system	Graduate Course Assignment	unspecified	beg	1
The IT Infrastructure Model	Information technology architecture	Graduate Course Assignment	unspecified	beg	1
The Requirements Ontology [35]	Software requirements	MSc Dissertation	kb application	adv	2
The Photography Model	Photography collection	Graduate Course Assignment	unspecified	beg	1
FIRA Ontology Refactored	Robot soccer matches	Graduate Course Assignment	onto analysis	adv	1
The Banking Model	Financial operations	Graduate Course Assignment	unspecified	adv	1
The Chartered Service Model	Railway chartered service	Graduate Course Assignment	unspecified	adv	1
The Health Organization Model	Brazilian health organization	Graduate Course Assignment	unspecified	beg	1
The Bank Model 2	Financial operations	Graduate Course Assignment	unspecified	adv	1
The PROV Ontology Refactored4	Provenance information	Graduate Course Assignment	ontol. analysis	beg	1
Web Service Modeling Ontology Refactored5	eGovernment services	Graduate Course Assignment	ontol. analysis	beg	1
The Recommendation Model	Recommendations and norms	Graduate Course Assignment	kb application	beg	1
The Inventory System	World view of an inventory management system	Other	interoperability	adv	1
MPOG Ontology Draft [21]	Brazilian federal organizational structures	Government Project	reference model	beg	7
The Project Management Model	Project management	Graduate Course Assignment	unspecified	beg	1
The Construction Model	Construction	Graduate Course Assignment	unspecified	beg	1
The Stock Model	Stoke brokers	Graduate Course Assignment	unspecified	beg	1
The Real State Model	Real state	Graduate Course Assignment	unspecified	beg	1

Table 2 provides a structural overview of the repository. The reader can observe that there are in the repository conceptual models of variable sizes, from large ones (e.g. The MGIC Ontology, with 3800 classes and 1918 associations), to medium (e.g. OntoUML Org Ontology, containing 78 classes and 78 associations), to very small ones (e.g. The Chartered Services Model, comprising 11 classes and 14 associations). Furthermore, it provides information of the number of each of the basic (Onto)UML constructs used in the models, including datatypes (complex datatypes, enumerations and primitive types), generalizations, generalization sets (Gen.Set) and attributes in each of the model.

[3]OWL ontology used to create the OntoUML version available at: http://musicontology.com/
[4] OWL ontology used to create the OntoUML version available at: http://www.w3.org/TR/prov-o/
[5] OWL ontology used to create the OntoUML version available at: http://www.wsmo.org/

Table 2. Structural description of all models in the repository.

Model	Class	Datatype	Association	Generalization	Gen. Set	Attribute
The MGIC Ontology	3800	61	1918	3616	698	865
The G.805 Ontology	135	4	113	127	36	0
The G.805 Ontology 2.0	358	1	255	475	62	7
The G.800 Ontology	477	1	345	601	78	7
OntoEmerge	189	4	138	111	16	5
OntoUML Org Ontology (O3)	78	0	78	57	8	0
The ECG Ontology	49	0	65	31	0	0
Gi2MO Ontology Refactored	65	5	63	42	7	2
Internal Affairs Ontology Refactored	62	0	54	36	9	0
Open proVenance Ontology	49	0	50	26	4	0
The Library Model	43	0	45	14	0	0
OntoBio	187	5	50	160	22	14
The Public Tenders Model	84	0	43	48	6	18
UFO-S	22	0	42	4	0	0
The TM Forum Model	34	0	41	20	4	0
The Social Contract Model	20	0	15	16	0	2
The Clergy Model	29	0	34	16	0	0
The FIFA Football Model	68	1	32	69	4	2
The PAS 77:2006 Ontology	66	0	32	55	11	0
IDAF Model	46	0	32	38	0	0
The Cloud Vunerability Ontology	33	0	29	21	2	0
The University Model	27	4	29	16	0	0
CMTO	41	0	28	28	0	0
GRU MPS.BR Model	19	7	28	15	3	18
The Experiment Model	20	2	26	0	0	0
CSHG Refactored	19	0	22	10	1	0
The Normative Acts Ontology	63	1	21	55	17	24
The Parking Lot System	49	0	21	37	9	17
The School Transportation Model	33	0	36	9	0	0
The Quality Assurance Ontology	41	0	20	24	7	2
The OpenFlow Ontology	20	0	19	9	1	4
The Music Ontology Refactored	43	0	18	36	6	5
The Internship Model	26	6	18	19	4	2
The G.809 Model	24	0	18	12	4	0
The ERP System Model	38	0	16	25	1	43
The Online Mentoring Model	29	0	16	18	6	0
The Help Desk System Model	20	0	16	8	4	0
The IT Infrastructure Model	31	0	15	17	6	0
The Requirements Ontology	35	1	21	30	10	19
The Photography Model	19	0	15	8	0	0
FIRA Ontology Refactored	41	0	14	36	7	0
The Bank Model	18	0	12	14	4	2
The Chartered Service Model	11	0	14	0	0	0
The Health Organization Model	24	0	13	14	3	0
The Bank Model 2	24	1	14	16	3	3
The PROV Ontology Refactored	16	0	12	5	0	0

WSMO Refactored	12	0	12	2	0	0
The Rec. Model	16	0	10	11	3	6
The Inventory System	20	0	7	14	0	24
MPOG Ontology Draft	15	0	7	15	4	0
The Project Management Model	14	0	7	8	3	0
The Construction Model	13	0	7	7	0	0
The Stock Model	14	0	6	11	7	0
The Real State Model	15	0	5	13	0	0

In order to analyze this new benchmark, we have implemented a set of computational strategies to automatically detect occurrences of these anti-patterns in OntoUML models (see discussion in Section 5). By running these algorithms for our initial set of anti-patterns under this benchmark, we managed to refine and extend the initial set elicited in [17] to a refined set of anti-patterns. **Table 3** reports on the results of this second study, which allowed us to refine our anti-pattern catalogue. The anti-pattern named **Binary Relation Between Overlapping Types (BinOver)** is a refinement and combination of the previous *Self-Type Relationship* and *Relation Between Overlapping Subtypes* anti-patterns identified in the first study. Moreover, we elicited two additional anti-patterns in this second study, namely **Repeatable Relator Instances (RepRel)**[6] and **Relator Mediating Overlapping Types (RelOver)**[7]. Furthermore, the **Association Cycle (AssCyC)** is the anti-pattern previously called *Generic Cycle*. Finally, we have excluded the *Pseudo-AntiRigid (PAR)* anti-pattern, identified in our original catalog, from the analysis conducted in this study. We did it because it would not be possible to specify an algorithm to identify its occurrences automatically.

Table 3 summarizes the results of our second study. It shows the total number of occurrences of each of these anti-patterns in the entire repository of 54 models. Moreover, it also shows what we term here the *qualified percentage of models with an anti-pattern occurrence*. In order to understand this metric, we first call attention to the fact that each anti-pattern requires the presence of certain modeling constructs in order for it to be manifested. For instance, the RelOver pattern requires the presence of Relators in order to be manifested. Therefore, the qualified occurrence of RelOver in the repository is the percentage of models in which this anti-pattern occurs within the populations of models in the repository that contain the presence of the *relevant model construct* Relator. In other words, it is the percentage of models in which the anti-pattern occurs given the population of models in which it could possibly occur. Finally, **Table 3** shows what we term the *element/anti-pattern rate*. This rate provides an estimation of the number of elements required for a modeler to produce an anti-pattern. For example, for every 3.77 occurrences of a relator, we expect an occurrence of the RepRel anti-pattern. We can also see that half of these anti-patterns require less than five uses of their respective relevant element type for the anti-pattern to be manifested.

Table 3. Anti-Pattern frequency on models with required elements.

Anti-Patterns (AP)	AP Occurrences	Relevant Model Construct (RMC)	RMC /AP Ratio	% of Qualified Models with AP Occurrence
RelSpec	817	Association	4.92	48.15%
ImpAbs	758	Association	5.30	72.22%
AssCyc	1809	Association	2.22	92.59%
RelOver	149	Relator	8.08	25%
RepRel	319	Relator	3.77	64.58%
BinOver	224	Association	17.93	48.15%

It is important to highlight that given the size of this new set of models, unlike in our first study, we were not able to check for each occurrence of these anti-patterns (4076 occurrences!) whether they were always cases of model fragments that entailed unintended consequences. For this reason, in the analysis reported in Table 3, it is not the case that each occurrence of an anti-pattern in the model necessarily means an unintended occurrence of the corresponding model fragment. However, in our previous empirical study, we could observe a very strong correlation between the high occurrence of these anti-patterns as model fragments and cases in which they were identified as unintended. In fact, that is exactly why they were identified as anti-patterns (as opposed to purely syntactic constraints) in the first place. In Section 6, we report a third empirical study intended to measure the

[6] In [10], the Repeatable Relator Instances (RepRel) anti-pattern was entitled Twin Relator Instances (TRI)
[7] In [10], the Relator Mediating Overlapping Types (RelOver) anti-pattern was entitled Relator With Overlapping Roles (RWOR)

probability that an occurrence of an anti-pattern actually entails unintended consequences. Moreover, in the study reported there, we measure how often the refactoring plans provided in Section 4 are selected by the designers of that model to solve an occurrence an anti-pattern that in fact characterizes a mistake.

4. A Catalog of Ontological Anti-Patterns

In this section, we present our catalog of anti-patterns for ontology-driven conceptual modeling. In order to facilitate learning, usage and comparison of the anti-patterns, we describe them following a consistent format. Initially, we discuss each anti-pattern in natural language, presenting the structure that characterizes the anti-pattern and the reasons why it may cause modeling problems. In the sequence, we present a table that summarizes the most important information discussed. The template we use to describe these anti-pattern tables contains the following items:

- **Name:** uniquely identifies the anti-pattern and intends to convey a brief idea of its content.
- **Acronym:** a short name to facilitate the documentation and communication about the anti-pattern.
- **Description:** a natural language description of the generic structure that characterizes the anti-pattern. It also presents required constraints to characterize the anti-pattern occurrence, when necessary.
- **Justification:** a brief discussion of why modelers should scrutinize the model structure identified by the anti-pattern.
- **Type:** identifies the types of the anti-pattern, which indicates the type of problem the structure suggests. We here consider two types of anti-patterns: (i*) Logical Anti-Patterns*: related to cases of model overconstraining or model underconstraining; (ii) *Scope:* related to cases of models with missing or unnecessary constructs in the model.
- **Feature:** indicates the element of the OntoUML's meta-model that is in the relevant modeling construct for that anti-pattern (e.g., relator).
- **Structure:** formal description that characterizes an occurrence of the anti-pattern. These descriptions consist of *pattern roles*, *constraints* and a *diagrammatic generic example*. Note that some anti-patterns have one or more structures, which we call variations[8]. The reader should note that:
  - **Pattern Roles** describe the elements that participate in the anti-pattern, their possible stereotypes and cardinalities. We give each pattern role a proper name;
  - **Constraints** refer to logical expressions that must always be true to characterize an occurrence of the anti-pattern. Constraints can be general, involving multiple roles, or role-specific;
  - **Generic Example** graphically exemplifies the generic structure of the anti-pattern. When the anti-pattern has structural variations, we present multiple examples.
- **Refactoring Plan:** Every anti-pattern must define a set of refactoring plans. These plans define a sequence of actions that modify the model in order to fix the domain misrepresentation issue. Some plans may be mutually exclusive, if they cannot be performed in the context of a single occurrence, or complementary, if they can. Note that some plans are only applicable to certain variations of the anti-pattern (identified by the tag [conditional]). The refactoring plans are composed mainly by the following types of actions:
  - **Create Constraint (OCL):** indicate the definition of addition of OCL invariants or derivation rules (e.g. making explicit how a relation is derived or forbidding instances to relate in certain conditions);
  - **Modify Element (Mod):** indicate a change in a model element. The most frequent ones are stereotype changes (e.g. from Formal to Material or from Collective to Kind) and meta-property changes (e.g., change the *isReadOnly* meta-property from an association end from *false* to *true*);
  - **New Element (New):** indicate the creation of model elements;
  - **Delete Element (Del):** indicate the elimination of model elements.

It is important to emphasize that, for accessibility reasons, we chose to express our OCL constraints patterns in our refactoring plans in terms of the corresponding illustrative generic structure presented for that anti-pattern. The constraint patterns can be easily generalizable to generic structures of arbitrary sizes, provided that they preserve the distinguished characteristic of the anti-pattern at hand.

Finally, after detailing the aforementioned anti-patterns properties, we present anti-patterns **examples** encountered in concrete models from the model repository we discussed in Section 3. In order to show how to

[8]Some anti-patterns contain multiple structures because they generate the same type of problem and we can fix them using very similar actions.

analyze and refactor models using the anti-patterns, we discuss the presented examples and select an appropriate solution. Note that these examples are not included in the anti-pattern summary table.

4.1 Association Cycle (AssCyc)

The AssCyc anti-pattern occurs when an arbitrary number of classes are connected through the same number of relations in a way that composes a cycle (in the traditional graph theoretical sense). In other words, one can start navigating relations from any class in the cycle and arrive back to the starting point without going through the same relation and visiting the same class more than once (except for the first/last node). Notice that we intentionally use the term "relation" in a general sense, because we mean both associations (material relations, formal relations and parthood relations, etc.) and generalization relations.

We argue in favor of analyzing the structure identified by the AssCyc anti-pattern because it allows for two very characteristic instantiations: one in which there are cycles at the instance level, and another one in which there are not. Our empirical studies strongly indicated that usually, in a given model, only one of these instantiations patterns should be allowed.

Not all cycles, though, are occurrences of this anti-pattern. The first requirement is that two or more associations must compose the cycle. Cycles exclusively composed of generalizations are excluded, because they are either a syntactical mistake that should be eliminated (e.g., when all generalizations are in the same direction), or just regular hierarchies. Cycles with only one association are most likely a characterization of another anti-pattern, named BinOver (Section 4.2), while cycles with two associations could characterize occurrences of the RelSpec (Section 4.4) or RelOver (Section 4.5) anti-patterns. The second constraint that must hold is that an occurrence cannot be exactly a characterization of the *Relator-Material Relations Design Pattern* (see Section 2), i.e., it cannot be a case where one of the types is a relator that connects two other types by mediation relations, which are connected between themselves by a material relation. This is the case for a very simple reason: the derivation from the relator to the material relation imposes closed cycles at the instance level. For the same reason, Derivation relations are not considered as valid component members of the AssCyc cycle. Derivations are always used in the same way, i.e., connecting a relator class to a material relation, and thus, always forming cycles. The last constraint that must hold for a proper characterization of the AssCyc anti-pattern is that every association in the cycle must be intentional. This requirement is justified to eliminate cases in which the semantic variability (open or closed instance-level cycles) has already been addressed by the derivation rule(s) created by the modeler.

We propose three refactoring plans for AssCyc: first, to enforce the open cycle instantiation scenario at instance level through the specification of an OCL invariant; second, is an analogous solution to forbid instance level cycles; third, we set one of the associations as derived and its corresponding derivation OCL rule is specified.

The aforementioned properties of AssCyc are summarized in Table 4.

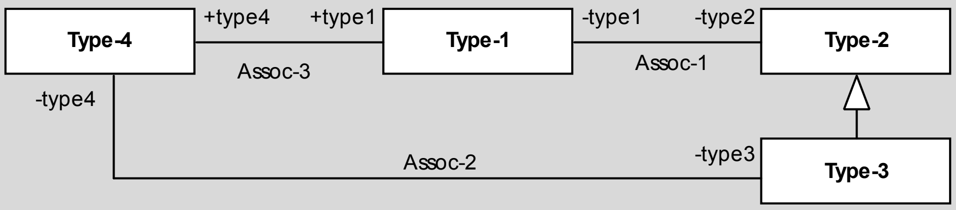
Table 4. Characterization of the AssCyc anti-pattern.

Name (Acronym)		**Description**
Association Cycle (AssCyc)		This anti-pattern occurs when n types are connected through n relations forming a cycle, i.e. one can start navigating relations from a type and arrive back without using the same relation more than once and without visiting the same type more than once (except for the first/last).
Type	**Feature**	**Justification**
Logical	Association	The analysis of two characteristic instantiation scenarios: one in which there are cycles at the instance level, and another one where there is not.
Pattern Roles		
Mult.	**Name**	**Possible Types**
3..*	Rel-n	Association (all but «derivation ») or Generalization
3..*	Type-n	Class

Constraints

1. The relations must form a cycle, i.e., starting from every Type$_n$, one can navigate the relations and arrive back to the same type using every relation exactly once.
2. The number of associations in the cycle must be greater than 2.
3. Every association must be primitive (*isDerived=false*)
4. The cycle cannot characterize a Relator Design Pattern.

Generic Example



Refactoring Plans

1. **[OCL] Enforce cycles:** create OCL invariant to enforce instance level cycles according to following template (any type in the cycle can be used as context):

```
context Type-1
inv: self.type2.oclAsType(Type-3).type4.asSet()->includes(self)
```

2. **[OCL] Forbid cycles:** create OCL invariant to forbid instance level cycles according to the following template (any type in the cycle can be used as context):

```
context Type-1
inv: self.type2.oclAsType(Type-3).type4.asSet()->excludes(self)
```

3. **[Mod/OCL] Derive association:** set the selected association as derived and create an OCL derivation rule. Suggested template bellow:

```
context Type-1::type1:Set(Type-4)
derive: self.type2.oclAsType(Type-3).type4->asSet()
```

To exemplify the AssCyc anti-pattern, consider the following fragment of the O3 ontology [28] depicted in **Figure 4**. The ontology describes a subset of the organizational domain. The most relevant concepts are: Formal Organization, such as a company or a university; Organizational Unit, which can be understood as departments of an organization; Employee Type, which captures the notion of what is commonly referred to as position, the official work post, like professor or manager; Business Role is a class that formalizes the idea of particular functions or roles, played by members of an organization. Examples are PhD supervisor and tutor. The relations state that an organization has two or more units; a unit defines roles that an employee can play; an organization defines positions to which people can be hired into; and that positions implied the possibility of playing certain business roles.

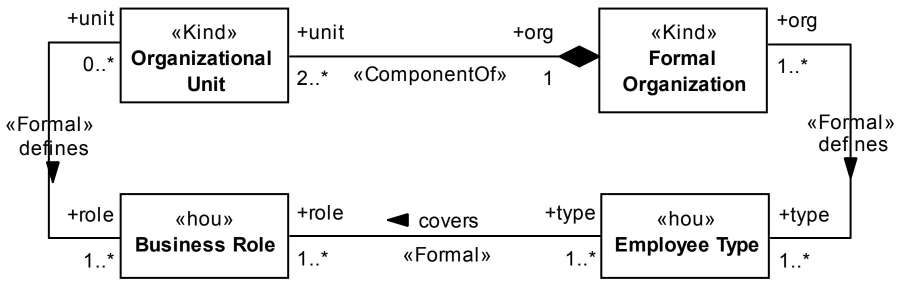


Figure 4. Fragment of the O3 ontology that contains an occurrence of the AssCyc anti-pattern.

As previously mentioned, the structure identified in **Figure 4** allows the instantiations of open and closed cycles at the instance-level. **Figure 5** presents a model instance automatically generated by the OLED's simulation component that exemplifies an open cycle. Notice that the organization defines an employee type (position) that covers business roles defined by organization units that are part of another organization. Alternatively, **Figure 6** presents an instantiation that characterizes a closed cycle. Note that, in this case, if a unit defines a business role, the organization it composes must define an employee type that covers such position.

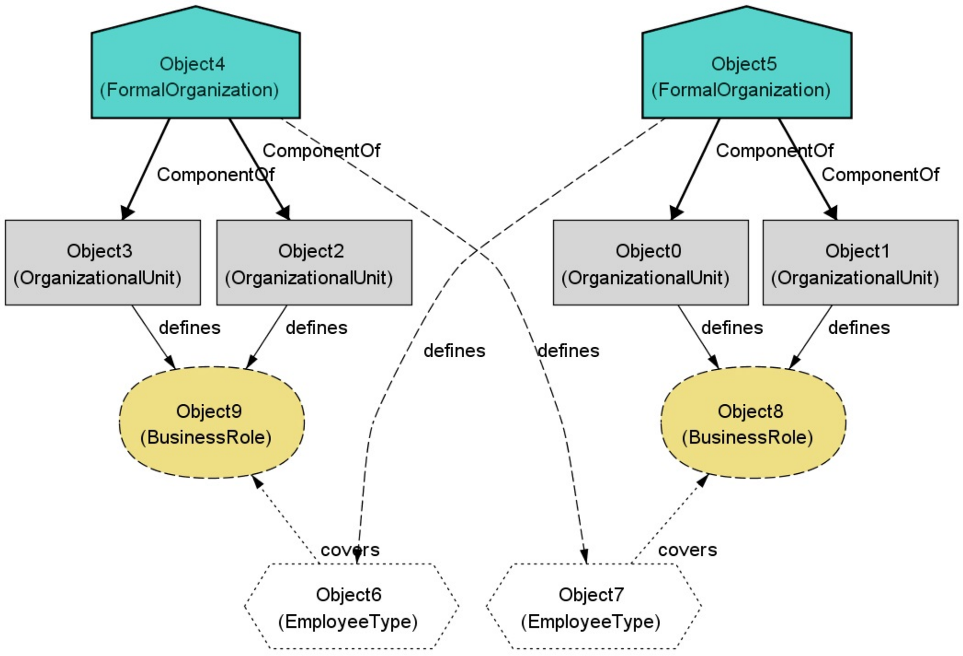


Figure 5. A possible instantiation of the O3 ontology, exemplifying an open instance cycle.

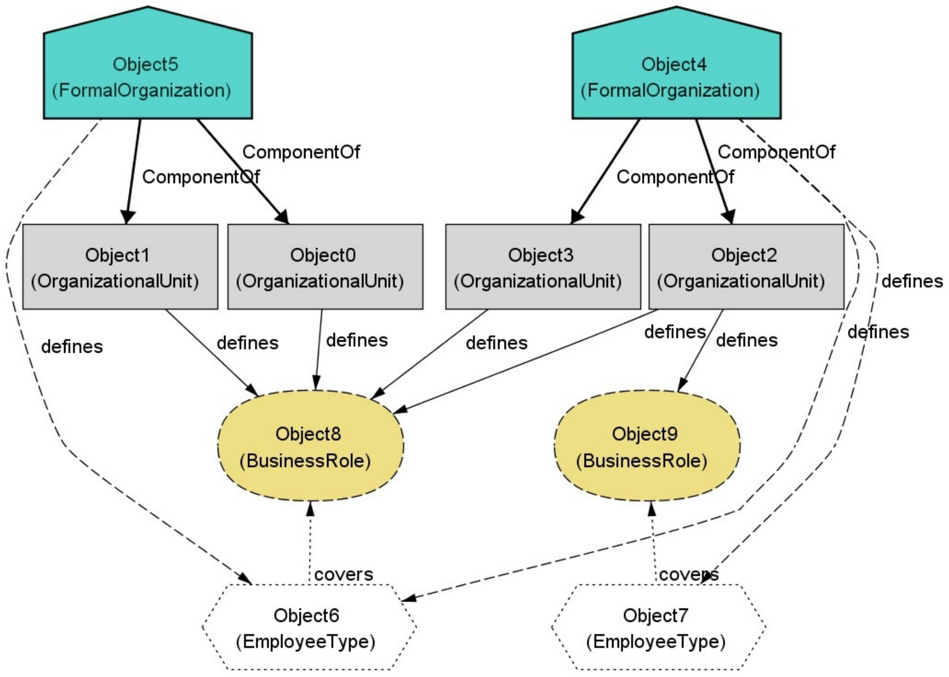
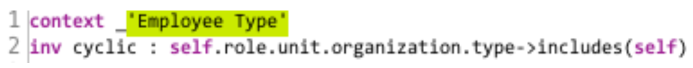


Figure 6. A possible instantiation of the O3 ontology, exemplifying a closed instance cycle.

When confronted with both situations, the authors of O3 decided that they only wanted their ontology to allow instances like the one in **Figure 6**. Since they also concluded that no association in the identified cycle was derived from the others, we proposed to rectify the model by adding the OCL invariant in **Listing 1**.



```
context _'Employee Type'
inv cyclic : self.role.unit.organization.type->includes(self)
```

Listing 1. OCL invariant generated to enforce closed cycles in the O3 fragment.

4.2 Binary Relation Between Overlapping Types (BinOver)

To describe this anti-pattern we first recall here the notions of an *overlapping* and *disjoint* types. Two types t and t' are said to **overlap** iff there is at least one possible instantiation of the model in which at least one individual simultaneously instantiate both types. For example, the set containing the Employee and Male is overlapping, since one can easily picture many examples of male workers in the world. We formally state this definition as follows:

<u>Definition 1 (Overlapping Types):</u> Let W be a non-empty set of possible worlds, $w \in W$ be a specific world, t and t' be particular types, $ext_w(t)$ the extension of t in world w and $exists(w)$ the function that maps a world w to all individuals that exists in it. Two types t and t' overlap iff:

$$\exists \boldsymbol{x}, \boldsymbol{w} \in W\ (\boldsymbol{x} \in \boldsymbol{exists}(\boldsymbol{w}) \land \boldsymbol{x} \in \boldsymbol{ext_w}(\boldsymbol{t}) \land \boldsymbol{x} \in \boldsymbol{ext_w}(\boldsymbol{t'}))$$

In contrast, two types t and t' are **disjoint** iff there is no possible instantiation in which an individual instantiate both types. As an example, consider the types Adult and Child. No individual, at any point of time, can instantiate all these types simultaneously.

<u>Definition 2 (Disjoint Types):</u> Making the same conventions as in the previous definition, two types t and t' are disjoint iff they do not overlap.

The Binary Relation Between Overlapping Types (BinOver) anti-pattern occurs when an association of any given meta-type (i.e., decorated with any OntoUML stereotype) connects two types that constitute an overlapping set. If this is the case, it means that the same individual may eventually instantiate both ends of the relationship. A given relation <R> between types <Source> and <Target> characterize a BinOver occurrence when:

1. <Source> equals <Target>
2. <Source> is a direct or indirect subtype of <Target>;
3. <Target> is a direct or indirect subtype of <Source>;
4. <Source> and <Target> are sortals (e.g., Subkind or Role) that share a common identity provider (e.g., Kind) and there is no generalization set which makes them explicitly disjoint;
5. <Source> and <Target> are relators that share a common supertype and there is no generalization set which makes them explicitly disjoint;
6. <Source> and <Target> are modes that share a common super-type and there is no generalization set which makes them explicitly disjoint;
7. <Source> and <Target> are mixins (e.g., RoleMixin) that directly or indirectly generalize at least one common sortal (e.g., Kind or Role);
8. <Source> and <Target> are mixins (e.g., RoleMixin) that share a common mixin super-type and none of their subtypes are sortals;

In a preliminary study, we reported structures (1) and (4) as being two different anti-patterns, named Self-Type Relationship (STR) and Binary Relation Between Overlapping Subtypes (BinOver), respectively [17]. After conducting further analysis, we realized that these structures should be merged and expanded into the anti-pattern presented in this section. The rationale is that, although they are different in structure, both anti-patterns convey the same conceptual problems and are amenable to the same set of solutions. It is important to precisely identify the structures (1-8), since it is not always intuitive that the related types overlap. When they do, we learned that is useful to specify binary relation meta-properties, like reflexivity or transitivity, in order to prevent misrepresentations of the domain. From our empirical studies, we learned that the most useful binary properties for conceptual modeling are reflexivity, symmetry, transitivity and cyclicity. We formally define each property in **Table 5**. For a detailed listing and description of more complex binary properties, please refer to [36].

The association's stereotype heavily influences the possible refactoring alternatives the modeler can apply to it. For some types of relations, such as characterizations, mediations and the different types of part-whole relations, the language already embeds binary properties, whilst on others, such as material and formal relations these constraints should be defined by the modeler when applicable. **Table 6** presents the embedded binary meta-properties values for the stereotypes of associations in OntoUML. In addition to the basic binary meta-properties, we identify in the last column, for each relation stereotype whether the language allows modelers to define type-reflexive associations of that given type.

Table 5. Relevant binary properties for conceptual modeling.

Binary Property	Definition	Example
Reflexive	$\forall x \in X \rightarrow R(x,x)$	is equal to
Antireflexive	$\forall x \in X \rightarrow \neg R(x,x)$	is father of
Symmetric	$\forall x,y \in X, R(x,y) \rightarrow R(y,x)$	is brother of, is married to
Antisymmetric	$\forall x,y \in X, R(x,y) \wedge R(y,x) \rightarrow x = y$	greater or equal to (≥)
Transitive	$\forall x,y,z \in X, R(x,y) \wedge R(y,z) \rightarrow R(x,z)$	is ancestor of
Acyclic	$\forall x_1, x_2, \dots, x_n \in X, R(x_1,x_2) \wedge R(x_2,x_3) \wedge \dots \wedge R(x_{n-1},x_n) \rightarrow \neg R(x_n,x_1)$	is ancestor of

Table 6. Binary property values embedded in OntoUML's associations.

Stereotype	Reflexivity	Symmetry	Transitivity	Cyclicity	Type-Reflexive
Formal	Undefined	Undefined	Undefined	Undefined	Allowed
Material	Undefined	Undefined	Undefined	Undefined	Allowed
Mediation	n.a.	n.a.	n.a.	n.a.	Forbidden
Characterization	Irreflexive	Asymmetric	n.a.	Acyclic	Forbidden
ComponentOf	Irreflexive	Asymmetric	Transitive	Acyclic	Allowed
MemberOf	Irreflexive	Asymmetric	Intransitive	Acyclic	Forbidden
SubCollectionOf	Irreflexive	Asymmetric	Transitive	Acyclic	Allowed
SubQuantityOf	Irreflexive	Asymmetric	Transitive	Acyclic	Forbidden

We propose three refactoring alternatives for a BinOver occurrence: change the association's stereotype, create OCL invariants to enforce a desired binary meta-property, or "force" the related types to be disjoint. Note that, formally speaking, if a modeler enforces the related types to be disjoint, she will not be able to set any binary meta-property, since the relation will no longer have the same individuals in the domain and range. **Table 7** summarizes the description of the BinOver anti-pattern.

In order to exemplify an occurrence of this anti-pattern, we extracted and adapted the small model fragment depicted in **Figure 7** from the MGIC Ontology [22]. This fragment depicts Railway Systems, i.e., collections of railways that the Brazilian government outsources to private organizations. These systems are connected to each other, which in the model is represented by the "isConnectedTo" formal association. This example fits the first structural configuration characterizing the BinOver anti-pattern, i.e., an association that connects a type to itself.

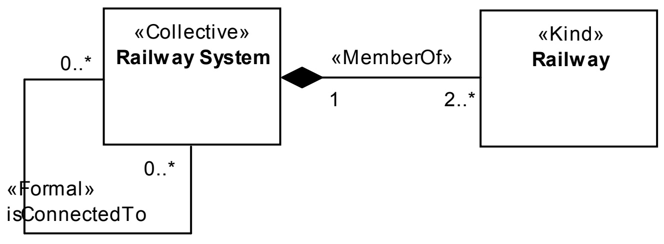


Figure 7. Fragment of the MGIC that exemplifies BinOver.

Figure 8 depicts two possible worlds (again, model instances automatically generated by the OntoUML tool). On the left, built using white boxes, is a representation of a possible world in which the relation of being connected is transitive and acyclic. On the right, composed by grey boxes, is a possible world in which the relation is both symmetric, reflexive, transitive and cyclic. In this case, the modelers have chosen to create a rule for making the relation symmetric and reflexive.

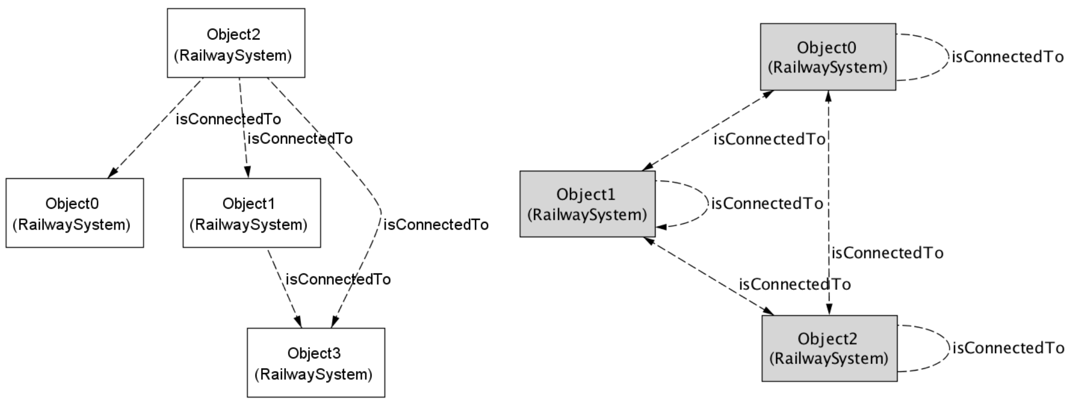


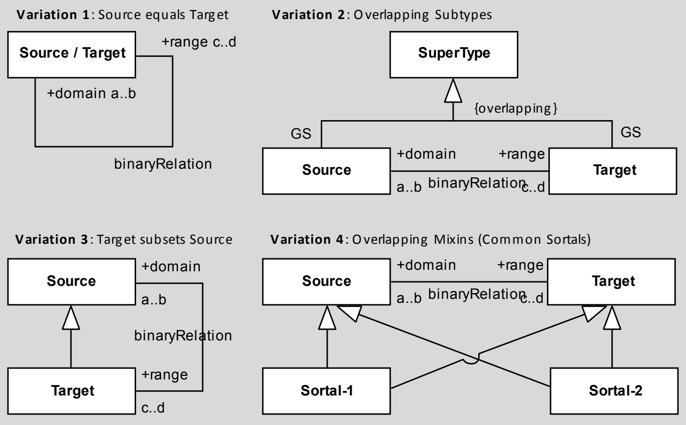
Figure 8. On the left, in white, a possible world in which the "isConnectedTo" relation is transitive and acyclic; on the right, in grey, a world where "isConnectedTo" is reflexive, symmetric and cyclic.

Table 7. Characterization of the BinOver anti-pattern.

Name (Acronym)		Description
Binary Relation between Overlapping Types (BinOver)		A binary relation whose end types are overlapping characterizes this anti-pattern.
Type	**Feature**	**Justification**
Logical	Association	Modelers often do not perceive by themselves that two or more types overlap. This anti-pattern makes them aware of that and confronts modelers with the possibility to specify binary relation meta-properties (e.g. reflexivity, transitivity and symmetry.

Pattern Roles		
Mult.	**Name**	**Possible Types**
1	binaryRelation	Association (all but «derivation »)
1	Source, Target	Class

Generic Example*



Refactoring Plans

1. **[Mod] Fix stereotype:** change the stereotype of the relation to fit a desired binary meta-property
2. **[OCL] Enforce binary property:** create OCL invariant to enforce a desired binary meta-property (as long as it is compatible with the embedded constraints of the stereotype)
3. **[New] Enforce disjointness:** make the related types disjoint by the specification of a disjoint generalization set.

4.3 Imprecise Abstraction (ImpAbs)

An association R characterizes the logical anti-pattern named **Imprecise Abstraction (ImpAbs)** if at least one of the following holds:

- R’s source end upper bound multiplicity is equal or greater than 2 and the Class connected to it has 2 or more subtypes;
- R’s target end upper bound multiplicity is equal or greater than 2 and the Class connected to it has 2 or more subtypes

Suppose we have an association R connecting two types T1 and T2. The source of the inconsistency in this case comes from the representation of a single, more abstract association between T1 and T2, instead of more concrete associations between the subtypes specializing T1 and T2. In such a situation, our empirical studies show that there can be hidden domain-specific constraints refering to, for instance, which subtypes of T2 an instance of T1 may be related. Moreover, the instances of T1 might be subject to different cardinality constraints on R for each of the different subtypes specializing T2. Finally, the implicit concrete specializations of R that should be defined between the subtypes of T1 and T2 can be subject to different values for meta-attributes such as *isDerived*, *isReadOnly*, *isEssential* and *isInseparable* [7].

This anti-pattern allows for three refactoring alternatives: (a) set cardinality constraints through the specification of an OCL invariant; (b) set cardinality constraints through the specification of a new association that subsets the original one; and (c) the specification of particular association meta-property values, also through the creation of an association.

Options (a) and (b) are equivalent in term of logical implications and, thus, are mutually exclusive for the same pair of classes. In constrast, alternative (c) can be combined with the first two, although if one is already going to create a new association, it is more reasonable to use it also to set the cardinality constraints. One should notice, nonetheless, that the constraints defined for relations between subtypes of the originally related classes cannot contradict the ones for the original relation. Minimum cardinalities must be lower or equal to the general relation’s minimum and maximum cardinality constraints. Maximum cardinalities must be greater than the general relation’s minimum cardinality and lower or equal to the general relation’s maximum carinality. Modelers can only customize other Boolean meta-properties, like *isEssential*, *isInseparable*, *isImmutableWhole*, *isImmutablePart*, *isShareable* [7,9] or *isReadOnly* if the value set for the generic relation has value *false*. Table 8 summarizes the characterization of the ImpAbs anti-pattern.

Table 8. Characterization summary of the ImpAbs anti-pattern.

Name (Acronym)		**Description**
Imprecise Abstraction (ImpAbs)		A given association R characterizes an ImpAbs occurrence if at least one of the following holds: (i) R’s source end upper bound multiplicity is equal or greater than 2 and the Class connected to it has 2 or more subtypes; (ii) R’s target end upper bound multiplicity is equal or greater than 2 and the Class connected to it has 2 or more subtypes
Type	**Feature**	**Justification**
Logical; Scope	Association	Representing merely a general relation between types T1 and T2 can causes the model to be too permissive given that there can be hidden domain-specific constraints referring to, for instance, which subtypes of T2 an instance of T1 may be related. Moreover, the instances of T1 might be subject to different cardinality constraints on R for each of the different subtypes specializing T2. Finally, the implicit concrete specializations of R that should be defined between the subtypes of T1 and T2 can be subject to different values for meta-attributes such as *isDerived*, *isReadOnly*, *isEssential* and *isInseparable*.
Pattern Roles		
Mult.	**Name**	**Possible Types**
1	Assoc	All association stereotypes

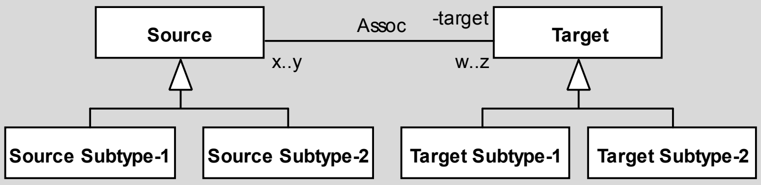
1	Source, Target	All class stereotypes
0..*	Source Subtype-n, Target Subtype-n	All class stereotypes

Additional Constraints

1. Let allSubtypes(c) be the function that return all direct and indirect subtypes of a class c, sourceEnd(a) and targetEnd(a) the functions that return the source and target ends of an association a, and upper(p) be the function that return the upper bound cardinality of a property p, then:

$$\left(upper\left(sourceEnd(Assoc)\right) \geq 2 \wedge \#allSubtypes(Source) \geq 2\right) \vee \left(upper\left(targetEnd(Assoc)\right) \geq 2 \wedge \#allSubtypes(Target) \geq 2\right)$$

Generic Example



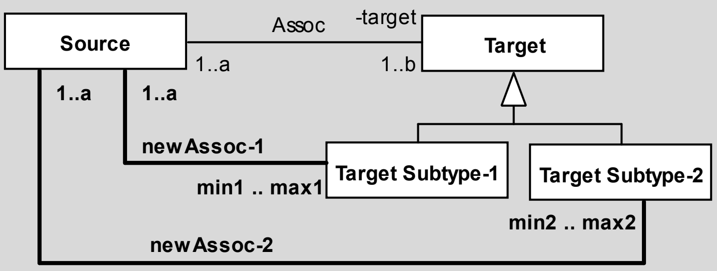
Refactoring Plans

1. **[OCL] Add multiplicity constraint:** choose this option if there is a domain restriction that requires an instance of Source, or of one of its subtypes, to be connected to a minimum, maximum or precise number of instances of Target, or one of its subtypes. The following OCL invariant enforces the desired constraint:

```
context Source
inv: let sub1Size =
self.target->select( x |x.oclIsTypeOf(_'Target Subtype-1'))->size()
   in sub1Size >= min1 and sub1Size <= max1
```

2. **[New] Add custom meta-property (subsetting association):** choose this option if the relation between Source and Target have particular meta-properties (like isReadOnly and isEssential) when an instance of Source, or of one of its subtypes, to be connected to a minimum, maximum or precise number of instances of Target, or one of its subtypes

3. **[New] Add multiplicity constraint (subsetting association):** this option has the same logical result of the first one. However, the results are achieved through the specification of a new association (using the same stereotype of Assoc) that subsets Assoc and whose cardinalities enforce the cardinality constraints.



As a concrete example of this anti-pattern, consider the occurrence identified in the Electrocardiogram (ECG) ontology [20] and depicted in **Figure 9**. The fragment states that a Heart contains atriums, ventricles and cells. The *componentOf* relation between Heart and Heart Cell induces a case of an ImpAbs occurrence. According to this model, it is possible for a Heart to exist containing only Non-Pacemaker Cells. However, this is an obvious unintended consequence since pacemaker cells are the ones responsible for the heart's contraction, i.e. a heart without them would not beat. To solve this problem, we need to create additional sub-relations, one for each type of required cell.

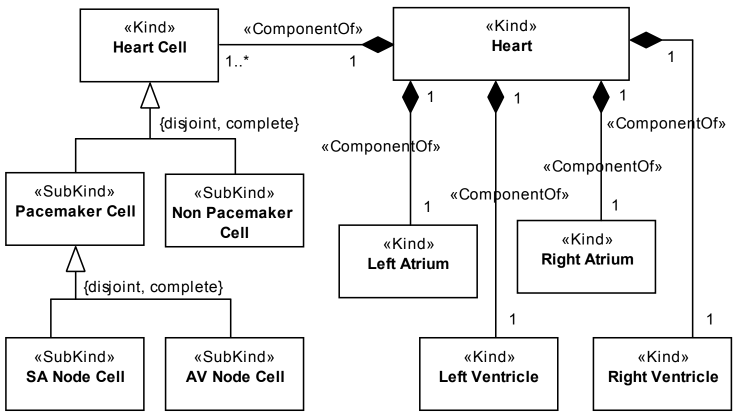


Figure 9. ImpAbs occurrence identified in the ECG Ontology.

As a last remark, we highlight that the empirical study discussed in Section 6 indicated that ImpAbs is more likely to be a problem when the association that characterizes the occurrence is a part-whole relation (with exception of the *memberOf* relation), like in the case of Figure 9. This reinforces the importance of modeling language that makes explicit the distinction between part-whole relations as well as the distinction between parthood and non-parthood relations.

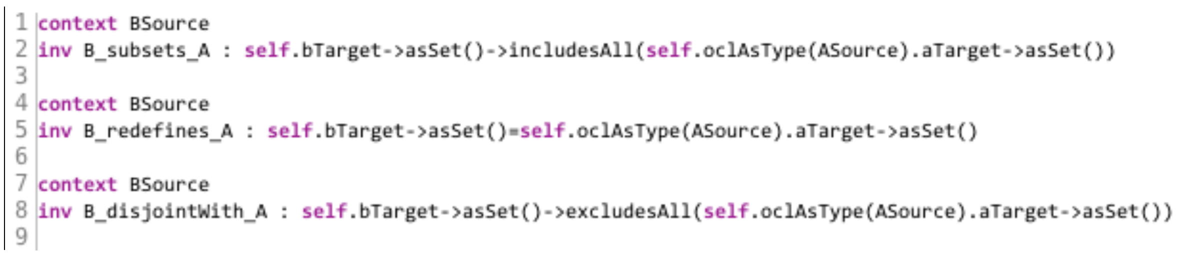
4.4 Relation Specialization (RelSpec)

The **Relation Specialization Anti-Pattern (RelSpec)** consists of two relations A and B that connect types ASource and ATarget, and BSource and BTarget, respectively, such that one of the following conditions holds:

- ASource is equal to or a subtype of BSource, and ATarget is equal to or a subtype of BTarget;
- ASource is equal to or a subtype of BTarget, and ATarget is equal to or a subtype of BSource

Our empirical studies showed the structures identified by this anti-pattern are likely to require additional constraints to specify a sort of dependency between the instantiation of associations A and B. We identified that often modelers needed to include one of the following four restrictions: *association subsetting*, *association redefinition*, *association disjointness* and *association specialization*. We go through each of these constraints individually in the following paragraphs.

We start with *association subsetting*. It is true that association B subsets association A if, and only if, being related through B implies being related through A but not the other way around. As an example, consider the relations of being a father and of being an ancestor, both which hold between people. It is true that "father of" subsets "ancestor of" because every father is an ancestor, but not every ancestor is a father. In other words, subsetting here means proper subsetting, If the *RelSpec* occurrence requires a *subsetting constraint*, the modeler should add one of the A's association ends, the subsetted association, to the subsetted properties list of the respective association end of B. The formal semantics of a subsetted property is described in [16] as well as in lines 1 and 2 of **Listing 2**.


```
1 context BSource
2 inv B_subsets_A : self.bTarget->asSet()->includesAll(self.oclAsType(ASource).aTarget->asSet())
3
4 context BSource
5 inv B_redefines_A : self.bTarget->asSet()=self.oclAsType(ASource).aTarget->asSet()
6
7 context BSource
8 inv B_disjointWith_A : self.bTarget->asSet()->excludesAll(self.oclAsType(ASource).aTarget->asSet())
9
```

Listing 2. Formal semantics of subsetting, redefinition and disjointness constraint written in OCL.

The second type of constraint is *association redefinition*. Association B redefines association A if, and only if, whenever an individual instantiates BSource, the individuals it is related through B are the same individuals it is related through A. Note that, like subsetting, in redefinitions, being related through B also implies being related through A. The difference is that there cannot be individuals related through the "parent" association but not through the “child” association. Analogous to subsetting, OntoUML’s meta-model specifies a list of redefined properties for each association end. The formal semantics of a redefined property is also defined in [16], as specified on lines 4-5 of **Listing 2**. We make an exception for adopting the redefinition constraint when A and B relate the same types. In these cases, the extension of the associations will always be same and they will turn out to be redundant relations, increasing the model’s complexity without providing new knowledge. In that case, the modeler can take two alternative paths: delete one of the associations and forget about the redefinition constraint; or specialize at least one of B’s end and keep the redefinition constraint.

The third type of constraint usually needed when analyzing a RelSpec occurrence is *association disjointness*. In this case, B is disjoint from A if, and only if, being related through B implies not being related through A. To exemplify, consider a queue and the relations of predecessor and successor, which hold between individuals in the queue. If an individual is the predecessor of another, it implies that it is not its successor. The OntoUML meta-model does not consider the possibility of disjoint relations. For that reason, to enforce a constraint of such nature, one should include the OCL invariant presented in lines 7-8 of Listing 4.

The last refactoring plan for the RelSpec anti-pattern is to make B a specialization of A. As showed in [16], specializing and subsetting have the same formal semantics, the inclusion constraint of B in A. However, specialization represents an intentional relation between types, i.e., all properties of the general relation are inherited by the specializing relation. Furthermore, the event that establishes both relations is also the same. To specify this constraint, one just needs to create a generalization between the relations (from B to A). We consolidate the description, generic structure, the refactoring plans and other details of the RelSpec anti-pattern in **Table 9**.

Table 9. Characterization summary of the RelSpec anti-pattern.

Name (Acronym)	**Description**
Relation Specialization (RelSpec)	Two associations A, connecting ASource to ATarget, and B, connecting BSource to BTarget, such that: (i) ASource is equal or a subtype of BSource and ATarget is equal or a subtype of BTarget; or (ii) ASource is equal or a subtype of BTarget and ATarget is equal or a subtype of BSource

Type	**Feature**	**Justification**
Logical	Association	The identified structure suggests the existence of a specialization between the relations or the need for including a subsetting, redefinition or disjoint constraint.

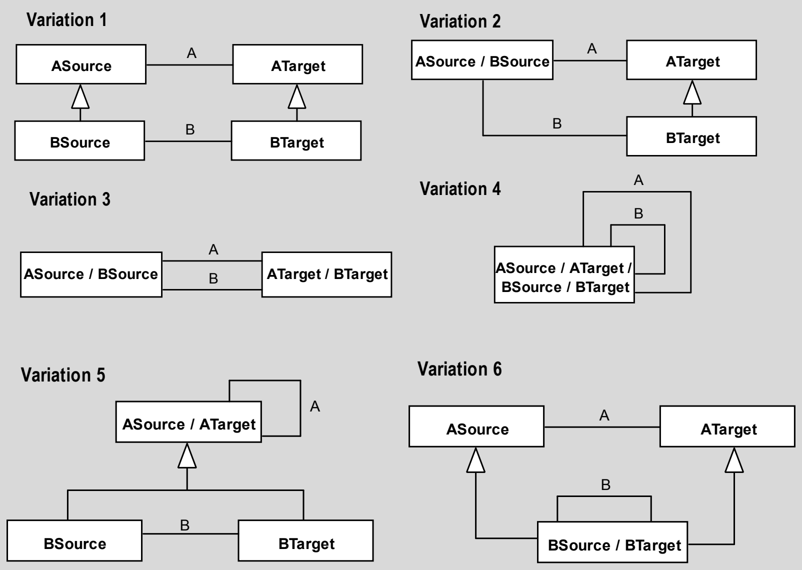
Pattern Roles

Mult.	**Name**	**Possible Types**
1	A, B	All association stereotypes
1	ASource, ATarget, BSource, BTarget	All class stereotypes

Additional Constraints

1. A and B are different associations
2. One of the following sentences must evaluate to true:
   $(ASource = BSource \lor ancestorOf(ASource, BSource)) \land (ATarget = BTarget \lor ancestorOf(ATarget, BTarget))$
   $(ASource = BTarget \lor ancestorOf(ASource, BTarget)) \land (ATarget = BSource \lor ancestorOf(ATarget, BSource))$

Generic Example*



**Note*: the presented variations are illustrative and do not intend to cover all possibilities*

Refactoring Plans

1. **[Mod] Subset:** take this solution if being connected through relation B implies being connected through relation A but not the other way around. To apply this solution, include in the meta-attribute *subsettedProperty* (a list) of one of B's association end the respective A end. Alternatively, the following OCL can be included in the model*:

```
context BSource
inv B_subsets_A : self.bTarget->asSet()->includesAll(self.oclAsType(ASource).aTarget-
>asSet())
```

2. **[Mod] Redefine:** this action should be taken if being related through B implies being related through A and also requiring that all related elements through A are related through B. To apply this solution, include in the meta-attribute *redefinedProperty* (a list) of one of B's association end the respective A end. Alternatively, the following OCL can be included in the model**:

```
context BSource
inv B_redefines_A : self.bTarget->asSet()=self.oclAsType(ASource).aTarget->asSet()
```

3. **[Mod/New] Disjoint:** this action should be taken if being related through B implies not being related through A.

```
context BSource
inv B_disjointWith_A :
self.bTarget->asSet()->excludesAll(self.oclAsType(ASource).aTarget->asSet())
```

4. **[New] Specialize:** the logical implication of this solution is the same as reinforcing subsetting. Nonetheless, it should only be selected if association B is truly a particular subtype of A (in the sense discussed in [16]) as opposed to the situation where a mere logical constraint is required between the extension of the two types.

*** This solution is strongly discouraged if associations A and B relate the same types.*

Figure 10 depicts a RelSpec occurrence identified in the OntoBio ontology [19]. The diagram presents the different relations between the concepts "*Environment*" and "*Spatial Location*". An "*Environment*" provides the biological characteristics (e.g., like vegetation, soil composition and climate) for a region delimited by geographical coordinates (defined by latitude, longitude and altitude). If a single coordinate defines a location, the authors named it a "*Geographic Point*". Furthermore, a *"Micro Environment" characterizes* it. Analogously, multiple coordinates (a region) define a "*Geographic Space*" that a "*Macro Environment*" characterizes.

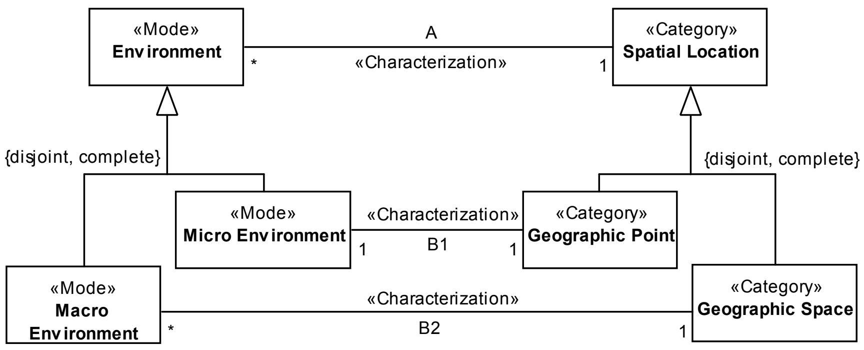


Figure 10. RelSpec occurrence identified in the OntoBio ontology.

In this diagram, we find two RelSpec occurrences: one composed of the characterizations A and B1 and another by characterizations A and B2. Now, we present possible instantiation allowed by models restricted using each of the constraints. For simplicity reasons, we only demonstrate scenarios using characterizations A and B2. The possible instantiation depicted in **Figure 11** exemplifies the implications of the subsetting constraint. As intended, the inclusion constraint of B2 in A is present: for all macro environments that the individual named "Object" is connected through B2, it is connected through A. However, "Object" is connected to "Property0" only though A, since the inclusion constraint at hand does apply both ways.

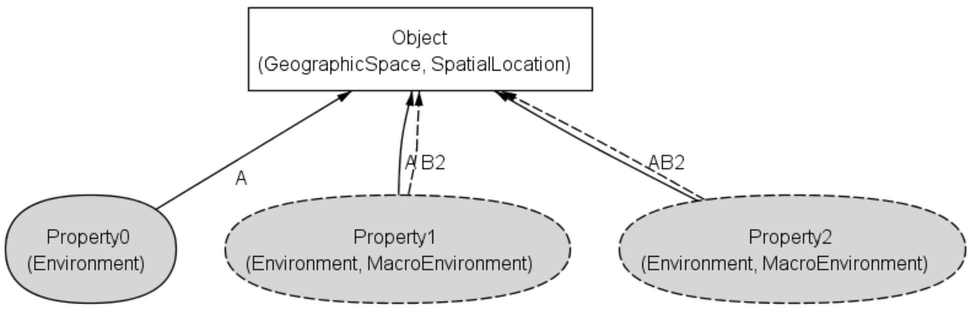

Object
(GeographicSpace, SpatialLocation)
A
A B2
AB2
Property0
(Environment)
Property1
(Environment, MacroEnvironment)
Property2
(Environment, MacroEnvironment)

Figure 11. Characterization of the subsetting constraint.

Figure 12 presents a valid world in case B2 redefines A. Note that, "*Object1*", which is an instance of "*Geographic Space",* relates to the same individuals through relations B2 and A. "*Object0*", in contrast, is not an instance of "*Geographic Space*", only of "*Spatial Location*", and thus the same restriction does not apply to it.

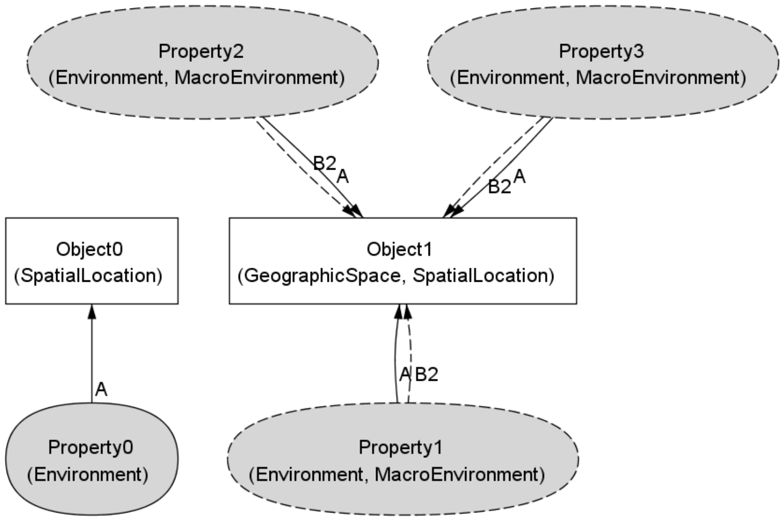

Property2
(Environment, MacroEnvironment)
Property3
(Environment, MacroEnvironment)
B2
A
B2
A
Object0
(SpatialLocation)
Object1
(GeographicSpace, SpatialLocation)
A
A B2
Property0
(Environment)
Property1
(Environment, MacroEnvironment)

Figure 12. Characterization of the redefinition constraint.

Finally, we present an exemplification of enforcing the disjoint constraint in **Figure 13.** Differently from the other scenarios, whenever a "Geographic Space" is connected to a "Macro Environment" through B2 it is not connected through A. "Object1", for example, is characterized by the environments "Property0" and "Property1" through A and by "Property2" through B2.

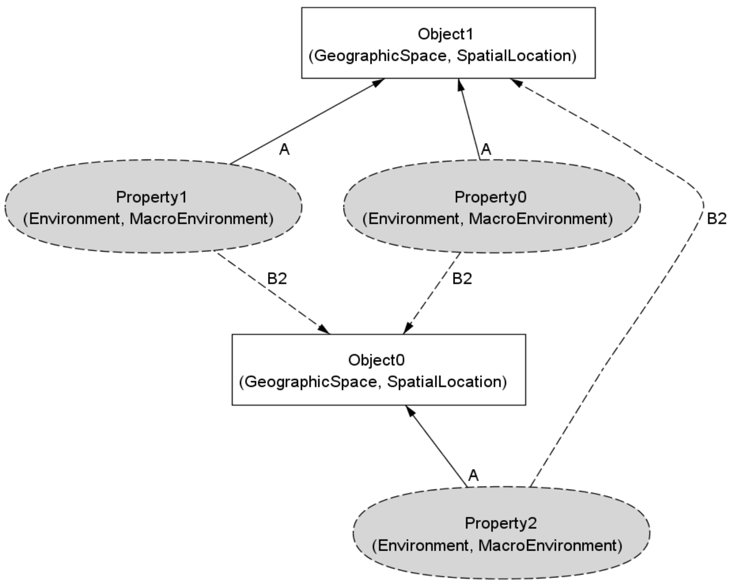

Object1
(GeographicSpace, SpatialLocation)
A
A
B2
Property1
(Environment, MacroEnvironment)
Property0
(Environment, MacroEnvironment)
B2
B2
Object0
(GeographicSpace, SpatialLocation)
A
Property2
(Environment, MacroEnvironment)

Figure 13. Characterization of the disjointness constraint.

The adopted solution for this RelSpec occurrence in the OntoBio ontology was to enforce the *redefinition* constraint on both relations B1 and B2.

4.5 Relator Mediating Overlapping Types (RelOver)

The **Relator Mediating Overlapping Types (RelOver)** is another logical anti-pattern. A relator connected to one or more sets of overlapping types[9] *mediation associations* characterizes an occurrence of this anti-pattern. In addition, the sum of the mediations' upper bound cardinalities on the mediated end (opposite to the end connected to the relator) must be greater or equal to 2. This is required to reduce the number of "false positives" since, according to OntoUML's syntactic constraints, every relator instance must mediate at least two distinct individuals [7]. This modeling structure is prone to be overly permissive, since there is no restriction for an instance to act as instances of multiples types for the same relator. The possible commonly identified intended interpretations are that:

- the mediated types are actually **disjoint**, i.e., regardless of the relator, there is no individual that can even instantiate more than one of the mediated types;
- all mediated types are **exclusive**, i.e. objects can simultaneously instantiate more than one mediated type, but what they cannot do is play more than one role in the context of the same relator instance; and
- **partially exclusive** mediated types, a weaker version of the previous alternative, in which some roles can be simultaneously played, whilst others cannot.

Table 10 summarizes the characterization of the RelOver anti-pattern.

Table 10. Characterization summary of the RelOver anti-pattern.

Name (Acronym)	**Description**
Relator Mediating Overlapping Types (RelOver)	A relator connected, through mediations, to two or more types whose extension possibly overlap. The sum of the mediations' upper bound cardinalities of the mediated end must be greater than 2.

[9]The notion of *overlapping types* adopted here is the same as previously defined for the BinOver anti-pattern.

Type	Feature	Justification
Logical	Relator	This structure is usually too permissive. It is often the case that some of the mediated types should be disjoint or exclusive in the context of a single relator instance.

Pattern Roles

Mult.	Name	Possible Types
1	Relator	«relator»
2..*	med-n	«mediation»
2..*	Over-n	All object types (e.g., «kind», «collective», «subkind», «role», «roleMixin»)

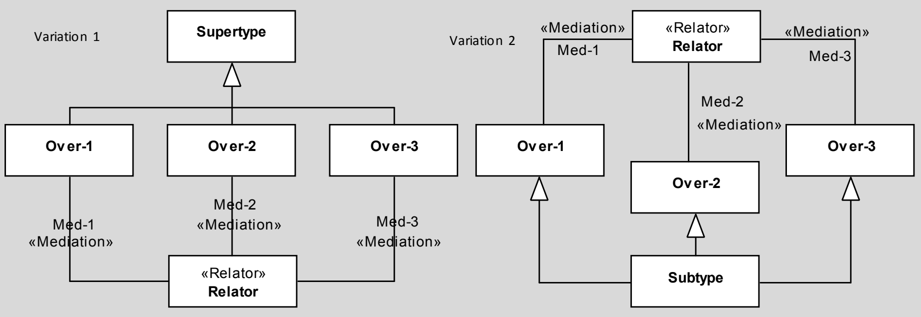
Additional Constraints

1. Let M be the set of identified mediations, mediatedEnd(m) the function that returns the association end opposed to relator of a mediation m, and upper(p) the function that return the upper bound cardinality of a property p, then:

$$\left(\sum_{m \in M} upper\big(mediatedEnd(m_n)\big)\right) > 2$$

2. Let O be the set of types mediated by a Relator, then there are at least two distinct types in O that overlap (see definition 2)

Generic Example*



*Note: the presented variations are illustrative and do not intend to cover all possibilities

Refactoring Plans

1. [OCL] Exclusiveness*: choose this option to forbid the same individual to play multiple roles w.r.t the same relator instance. Create an OCL invariant according to the following template:

```
context Relator
inv exclusiveTypes:self.over1->asSet()->excludesAll(self.over2->asSet()) and
                   self.over1->asSet()->excludesAll(self.over3->asSet()) and
                   self.over2->asSet()->excludesAll(self.over3->asSet())
```

2. [OCL] Partially exclusiveness: choose this option to forbid a subset of mediated types as exclusive.

3. [Mod/New] Disjoint mediated: Enforce types to be disjoint through the creation or alteration of a disjoint generalization set.

*Note: to make all types exclusive for n types, every binary combination should be pairwise explicitly ruled out

Figure 14 depicts a fragment of UFO-S, a commitment-based core reference ontology of services [24]. The fragment provides a partial description of the concepts of "*Service Offering*" and "*Service Agreement*". A provider makes an offering, which describes the terms in which she will provide the service. The agreement formalizes that a customer and a provider already negotiated the terms for hiring a service. The authors exemplify UFO-S using the car rental domain. A car rental company acts as a "*Service Provider*", when offering to rent cars. Their "*Target Customer* Community" contains, as members, all adults. When "Luke", for example, decides to rent a car from a particular company and signs the rental agreement, he acts as the "*Service Customer*" and the company as the

"*Hired Service Provider*". The rental agreement is the "*Service Agreement*", which specifies the conditions in which the service has been hired (price, duration, insurance and so on). The relator "*Service Agreement"* characterizes the RelOver occurrence, because it is the truth-maker of a material relation involving the overlapping types "*Hired Service Provider"* and "*Service Customer*. Note that, although the upper multiplicity on the provider side is equal to one, the upper bound multiplicity on the customer side is equal to many (*).

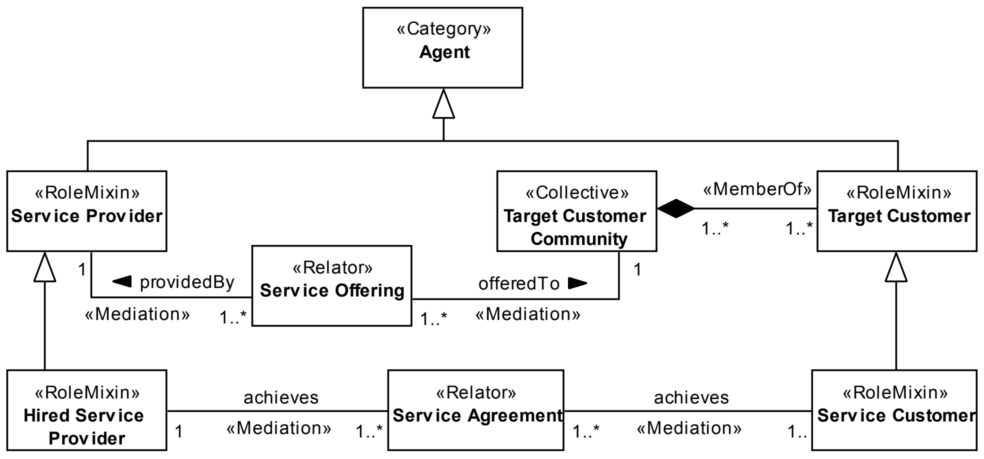

«Category» Agent
«RoleMixin» Service Provider
«Collective» Target Customer Community
«MemberOf»
1..*
1..*
«RoleMixin» Target Customer
1
◄ providedBy
«Relator» Service Offering
offeredTo ►
1
«Mediation»
1..*
1..*
«Mediation»
«RoleMixin» Hired Service Provider
achieves
«Relator» Service Agreement
achieves
«RoleMixin» Service Customer
1
«Mediation»
1..*
1..*
«Mediation»
1..

Figure 14. RelOver occurrence encountered in the UFO-S ontology.

The analysis of any RelOver occurrence starts by verifying whether the modeler intends the mediated types to be declared as disjoint. In our example, the inquiry is whether it is possible for a provider to also be a customer (e.g., a car rental company hiring the services of an accounting company). We assume here that the answer is yes, i.e. no disjointness constraint is required.

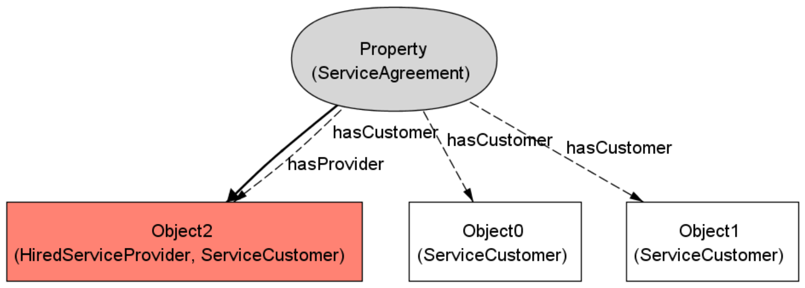
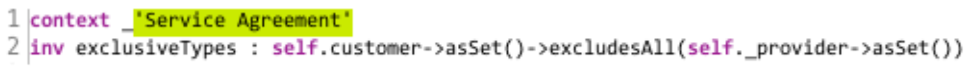

Property (ServiceAgreement)
hasCustomer
hasProvider
hasCustomer
hasCustomer
Object2 (HiredServiceProvider, ServiceCustomer)
Object0 (ServiceCustomer)
Object1 (ServiceCustomer)

Figure 15. Overlapping mediated types without exclusiveness constraint.

Keeping the mediated types as overlapping allows situations like the one depicted in **Figure 15**, i.e., a world in which one individual is both the provider and the customer in the context of the same agreement. If the authors decide to forbid such instantiations, they should enrich their ontology with the OCL invariant in **Listing 3**.



```
context _'Service Agreement'
inv exclusiveTypes : self.customer->asSet()->excludesAll(self._provider->asSet())
```

Listing 3. OCL invariant to enforce exclusive mediated types.

To complete our example, **Figure 16** shows a possible model instantiation still allowed after adding the exclusiveness constraint. "Object1" and "Object2" play both the provider and the customer roles, but now in the context of different agreements. "Object0", conversely, is just a customer in both agreements.

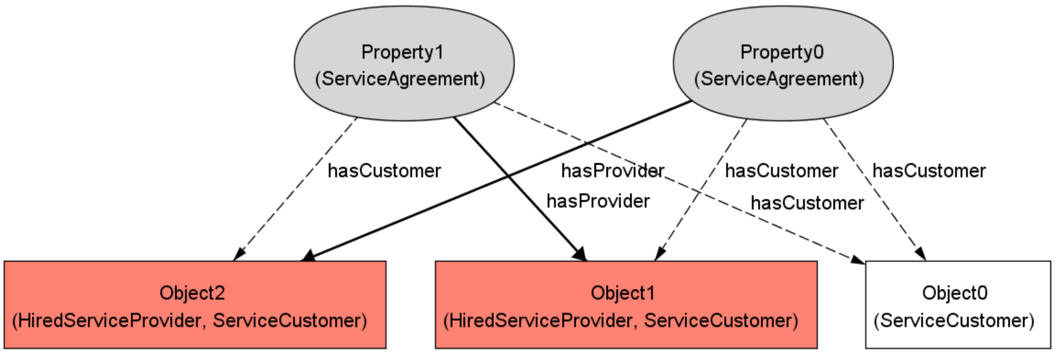
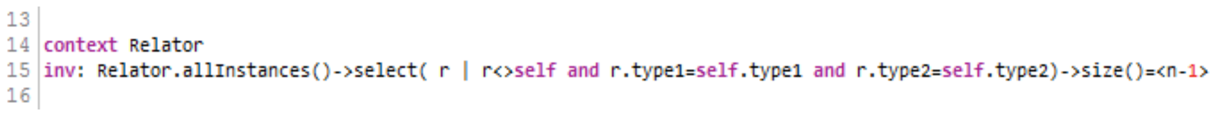


Figure 16. Simultaneous role instantiation with exclusive relators

4.6 Repeatable Relator Instances (RepRel)

The default semantics of an association is that of a set of tuples (as opposed to a bag of tuples). For example, if applied to the binary predicate "owns" in a model defined between the types Person and Car, it would forbid a person to own the same car more than once. Extensions for the "owns" predicate like {(John, Car1), (Joseph, Car2), (Luke, Car3)} would be accepted, but {(John, Car1), (Joseph, Car2), (John, Car1)} would not. In OntoUML models, relational properties are typically represented via the explicit reification of relator types and, in these models, in the absence of further constraints, the number of relator instances that mediate the very same set of relata could remain unconstrained. As our empirical studies show, the presence of relators in a model without these constraints constitutes a case of an anti-pattern that we termed the **Repeatable Relator Instances Anti-pattern (RepRel)**.

Moreover, we have noticed that, in using the language, modelers employ the following finer grained distinction required for specifying the limit of coexisting relator instances that mediate the exact same set of relata, namely, the distinction between *current* and *historical* semantics for relators. On one hand, a **current relator** is one whose instantiation corresponds to the *presence* of the individual in a given world, i.e., if a relator is instantiated in a world it is because it is present (causally active) in that world. On the other hand, **historical relators** are the ones that are intended to capture a record of the existence of a relator. This distinction is analogous to the distinction between Living and Deceased People in the model: the instances of both classes in a world w exist in that world. However, the instances of Living People are also present in that world. Typically, in an OntoUML model, people model this historical view on relators by defining phases such as "active" and "inactive" for relator types or by explicitly representing the lifetime of the relator (e.g., by defining temporal qualities for their moments of creation and termination). To exemplify the distinction between historical and current relators, consider the relator marriage (for simplicity, let us assume monogamous marriages in this example) connecting the roles husband and wife. In our considered normative system, it is only possible for a man to be married to exactly one wife at a time and vice-versa. Nonetheless, throughout one's life, one can marry again, if properly divorced. If this domain is modelled using current semantics for the relator, the cardinality would be exactly one on the relator end (and that would still allow many marriages throughout time). Conversely, if the modeler assumes a historical view, the multiplicity would be one or more on the relator end. To analyze an occurrence of the *RepRel* anti-pattern, a modeler must decide which of the aforementioned semantics she intends for the relator. If the semantics of current relators is adopted, adding the following OCL invariant in **Listing 4** will restrict the number of repeated instances.

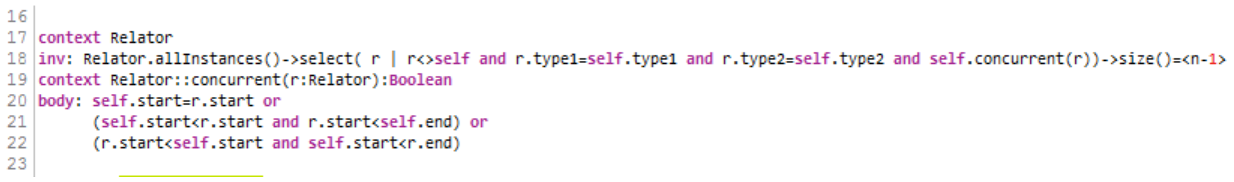


```
context Relator
inv: Relator.allInstances()->select( r | r<>self and r.type1=self.type1 and r.type2=self.type2)->size()=<n-1>
```

Listing 4. OCL version of the Uniqueness Constraint for "current" relators.

We should call attention for the fact that the internal uniqueness constraints can be directly expressed by the Relator-Material Relation pattern depicted in Fig. 1. By setting the variables η and λ in the association end connected to the relator type in the derivation relation, the modeler can precisely define how many relator instances can be associated with an n-uple of the material relations. However, we have also observed that, modelers frequently chose not to represent the material relation at hand, leaving the model only with the relator type and the corresponding mediation relations connecting it to the relata type (see an example in figure 17). This is specially the case, when the material relation at hand has an arity higher than 2 (i.e., ternary relations, quaternary relations, etc.). For this reason, we chose to include the constraint described in **Listing 4** as an alternative to the full specification of the Relator-Material Relation Pattern.

Enforcing uniqueness constraints on historical relators is more complex. We investigate here the case in which the modeler choses to explicitly represent the temporal interval over which a relator is supposed to exist via time stamped attributes such as “start”, to identify the creation time of the relator, and “end”, to identify its termination time. Since OntoUML does not specify a built-in datatype library, one would need to create her own Time datatype. In this situation, in order to eliminate a possibly unintended case of the RepRel anti-pattern, the modeler should add the OCL code provided in **Listing 5**.



```
context Relator
inv: Relator.allInstances()->select( r | r<>self and r.type1=self.type1 and r.type2=self.type2 and self.concurrent(r))->size()=<n-1>
context Relator::concurrent(r:Relator):Boolean
body: self.start=r.start or
      (self.start<r.start and r.start<self.end) or
      (r.start<self.start and self.start<r.end)
```

Listing 5. Enforcing Uniqueness Constraint for “historical” relators using OCL.

We provide a complete summary of the RepRel anti-pattern in **Table 11**.

Table 11. Characterization summary of the RepRel anti-pattern.

Name (Acronym)		**Description**
Repeatable Relator Instances (RepRel)		A «relator» connected to two or more «mediation» associations, whose upper bound cardinalities at the relator end are greater than one.
Type	**Feature**	**Justification**
Logical	Relator	This anti-pattern aids the modeler in specifying the number of different relators instances that can mediate the exact same set of individuals.

Pattern Roles		
Mult.	**Name**	**Possible Types**
1	Relator	«relator»
2..*	med-n	«mediation»
2..*	Type-n	«kind», «quantity», «collective», «subkind», «role», «phase», «roleMixin», «mixin» and «category»

Additional Constraints

1. Let M be the set of the mediations that characterize RepRel, relatorEnd(m) the function that return the association end whose type is the relator of a mediation m, and upper(p) the function that return the upper bound cardinality of a property p, then:

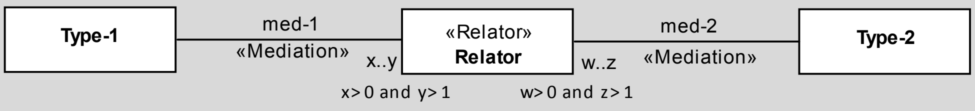
$$\forall m \in M, upper(relatorEnd(m)) > 1$$

2. Let M be the set of the mediations that characterize RepRel, relator (m) the function that returns the relator connected to a mediation m, and Relator the types stereotyped as «relator» in the RepRel anti-pattern, then:

$$\forall m \in M, (relator(m) = Relator) \vee (isAncestor(relator(m), Relator))$$

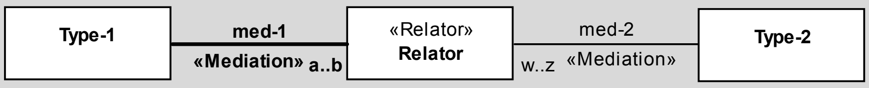
$$\exists m \in M, relator(m) = Relator$$

Generic Example



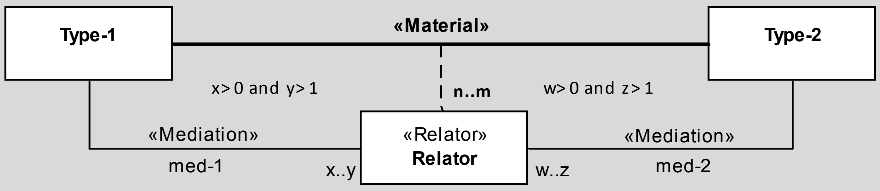
Refactoring Plans

1. **[Mod] Fix Multiplicites:** Set the lower and upper bound multiplicities for the relator end of a mediation relation to the appropriate values. Apply this solution at most once for each mediation relation connected to the possibly *repeatable relator*. Note that this plan will completely solve the issue highlighted by RepRel, only if at most one upper bound on the relator end of all mediations is greater than one.



Note: a and b are the new values assigned to the relator end of med-1.

2. **[OCL] Current Relator: add uniqueness constraint for a pair of types (Derivation):** Properly apply the Relator-Material Relations pattern [7] to limit the number of concurrent relator instances that mediate the very same set of individuals. First, if the model does not explicitly specify a material between the two types, create one, alongside a derivation relation connecting the new material association to the relator (type) at hand. Then use the lower and upper bound of the cardinality constraints at the relator end of the derivation relation to specify the minimum and maximum number of possible concurrent relator instances. This solution can be applied for each occurrence of the Relator-Material Relation Pattern. If the material relation at hand and, hence, the corresponding derivation relation have not been represented in the model, an alternative to this solution is alternative 3 below.



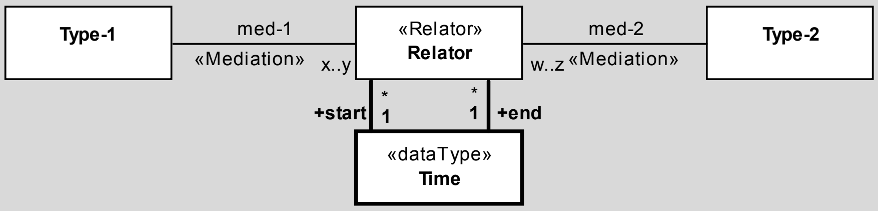
Note: <n> and <m> define the minimum and maximum number of relators instances mediating the same individuals.

3. **[OCL] Current Relator: add uniqueness constraint (Invariant):** Include an OCL invariant in the model to limit the number of concurrent relator instances that mediate the very same **set** of individuals, following the example below. Apply this solution as many times as needed for a given RepRel occurrence, but not more than once to the same set of types. Running this plan for two types has the same logical consequence as plan 2.

```
context Relator
inv: Relator.allInstances()->select( r | r<>self and r.type1=self.type1 and
r.type2=self.type2 and r.type3=self.type3)->size()=<n-1>
```

Note: the invariant exemplifies a uniqueness constraint between three types connected to the repeatable relator.

4. **[OCL] Historical Relator: add uniqueness constraint (Invariant):** This plan adopts a historical perspective on the existence of the relator to create an OCL invariant. First, create a Time datatype (if you do not have one in your model) and specify two properties for the repeatable relator – "start" and "end" – which you will use to define the time interval in which the relator instances are supposed to exist. Then add an operation named "concurrent" to the repeatable relator using the OCL expression below. Lastly, specify the uniqueness constraint using an OCL invariant, as also specified below. It is not recommended to apply this plan in combination with 2 and 3.



```
context Relator
inv: Relator.allInstances()->select( r | r<>self and r.type1=self.type1 and
r.type2=self.type2 and concurrent(self,r))->size()=<n-1>

context Relator::concurrent(r:Relator):Boolean
body: self.start=r.start or
   (self.start<r.start and r.start<self.end) or r.start<self.start and self.start<r.end)
```

Figure 17 depicts a RepRel occurrence identified in the Configuration Management Ontology (CMTO) [23]. The fragment focuses on the relator “Change Request”, which captures a registry of the action made by a “Requester”, when soliciting changes in one or more versions of a configuration item.

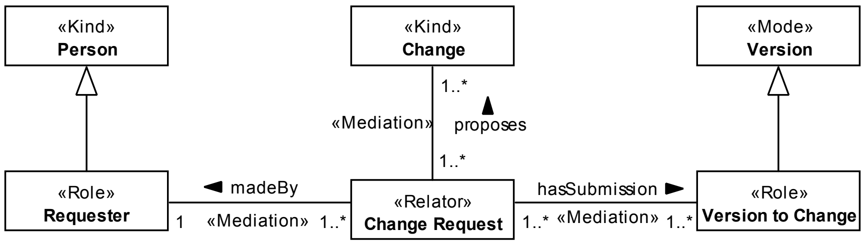


Figure 17. RepRel occurrence identified in the CMTO ontology.

The authors explicit use the word “register” when describing the relator class “Change Request”. Furthermore, they make an observation that there should be a quality attribute in the model to identify the request’s time of creation, even though they did not explicitly represented it. Considering these two issues along with the purpose of the ontology (i.e., provide support for the integration of configuration management system), we reasonably conclude that the authors intend a historical semantics for the relator “Change Request”. Continuing the analysis of this fragment, we use the simulation to generate examples and encounter the possibility depicted in **Figure 18**. As it can be noted, the ontology allows the same requester to make more than one request regarding the same change and the same version. If that is not desirable, one only needs to enrich the model with the OCL invariant described in **Listing 6**.

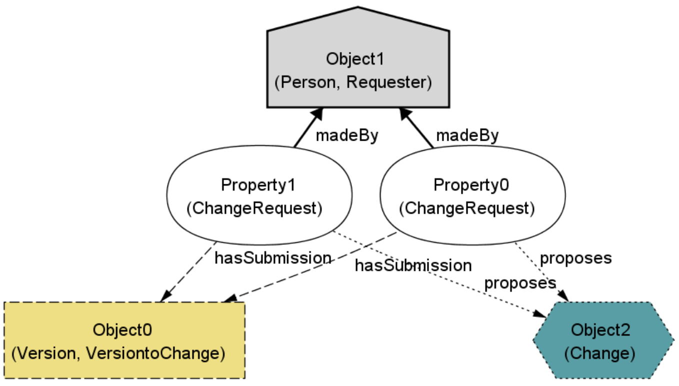
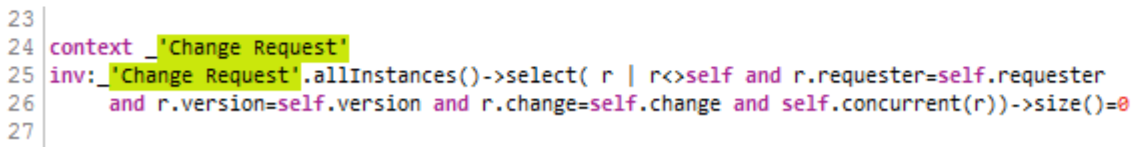


Figure 18. Possible world generated for the diagram in **Figure 17** without adding the uniqueness constraint regarding the relator *Change Request.*


```
context _'Change Request'
inv:_'Change Request'.allInstances()->select( r | r<>self and r.requester=self.requester
    and r.version=self.version and r.change=self.change and self.concurrent(r))->size()=0
```

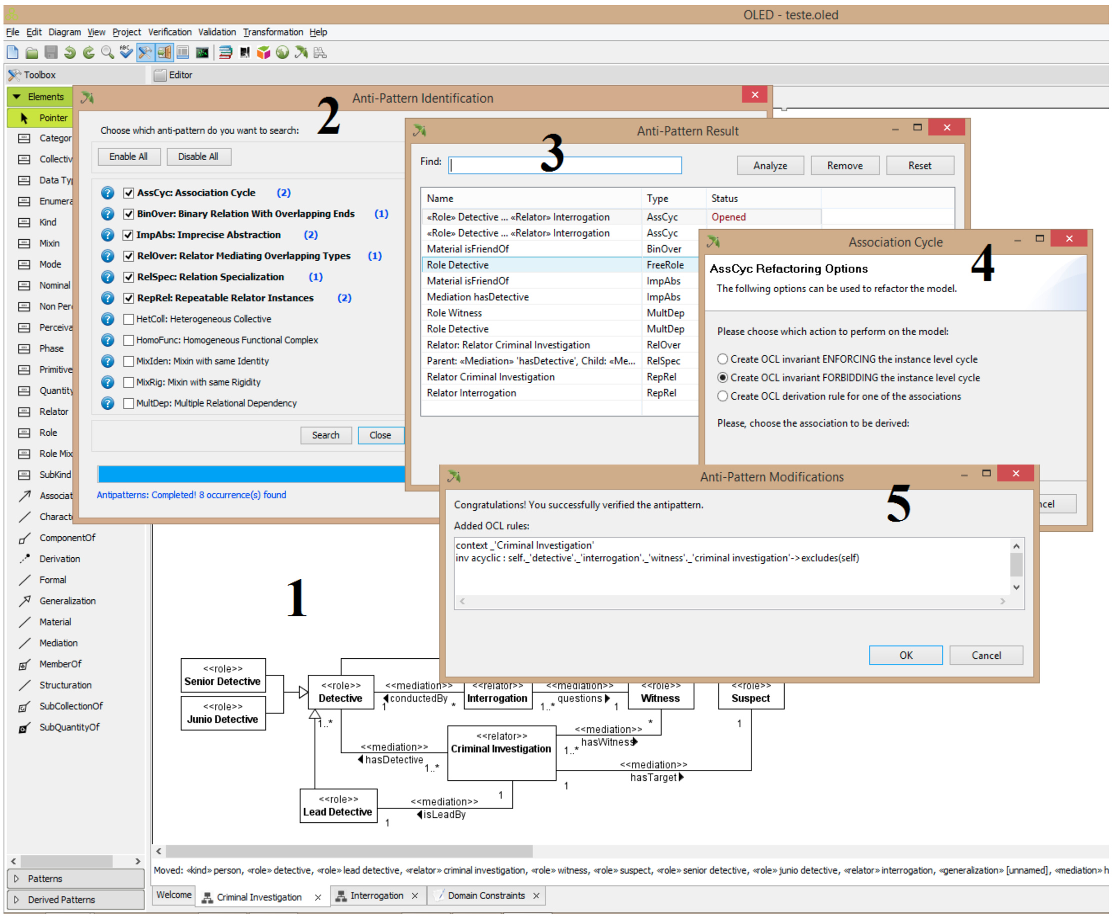
Listing 6. OCL constraint to limit repeated relators in the CMTO ontology

5. Anti-Pattern Occurrence Detection, Analysis and Elimination

In order to support the use of the anti-pattern catalog presented in the previous section, we developed a plug-in for the OntoUML Lightweight Editor (OLED) (**Figure 19**) [8]. The goal was to provide accessible alternatives for modelers to validate their ontologies without any additional training in special methods, tools or techniques. With that in mind, we adopted a simple strategy for managing anti-patterns that consists of three basic steps: automatic detection, guided analysis and automatic refactoring.

In order to relieve modelers from learning all anti-pattern structures and manually inspecting occurrences in their models, we implemented a component that does that automatically. Users can request an anti-pattern inspection on a particular diagram or on an arbitrary selection of elements. Moreover, as shown in **Figure 19.2**, users might instruct the tool to detect only a subset of the defined anti-patterns. In the sequence, an additional dialog lists all occurrences identified for the selected anti-patterns, as depicted in **Figure 19.3**. The tools lists the results using three columns: (i) “Name”, which corresponds to a short description of the most relevant elements that characterize the anti-pattern; (ii) “Type”, that provides the acronym for the anti-pattern type; and (iii) “Status”, a binary property that can be set as “Opened”, if the respective occurrence still has not been analyzed; and as “Fixed”, if the occurrence has been successfully analyzed.

The “Analyze” button gives rise to the second step of our strategy: the guided analysis. As we previously discussed, anti-patterns, in the sense that we use them, do not necessarily imply domain misrepresentations. In order to decide whether a particular occurrence entails unintended consequences, a modeler must reason about its consequences. To support this process, we implemented a wizard for each anti-pattern, which details the elements that participate in the anti-pattern occurrence, provides theoretical notions when necessary, and makes a series of questions, which lead to the appropriate solutions. It also provides a direct set of solutions for those users who are already familiar with the anti-pattern details, as shown in **Figure 19.4.**



Anti-Pattern Identification

Choose which anti-pattern do you want to search:

Enable All | Disable All

- [✓] **AssCyc: Association Cycle** (2)
- [✓] **BinOver: Binary Relation With Overlapping Ends** (1)
- [✓] **ImpAbs: Imprecise Abstraction** (2)
- [✓] **RelOver: Relator Mediating Overlapping Types** (1)
- [✓] **RelSpec: Relation Specialization** (1)
- [✓] **RepRel: Repeatable Relator Instances** (2)
- [ ] HetColl: Heterogeneous Collective
- [ ] HomoFunc: Homogeneous Functional Complex
- [ ] MixIden: Mixin with same Identity
- [ ] MixRig: Mixin with same Rigidity
- [ ] MultDep: Multiple Relational Dependency

Search | Close

Antipatterns: Completed! 8 occurrence(s) found

Anti-Pattern Result

Find:

Analyze | Remove | Reset

Name	Type	Status
«Role» Detective ... «Relator» Interrogation	AssCyc	Opened
«Role» Detective ... «Relator» Interrogation	AssCyc	
Material isFriendOf	BinOver	
Role Detective	FreeRole	
Material isFriendOf	ImpAbs	
Mediation hasDetective	ImpAbs	
Role Witness	MultDep	
Role Detective	MultDep	
Relator: Relator Criminal Investigation	RelOver	
Parent: «Mediation» 'hasDetective', Child: «Me...	RelSpec	
Relator Criminal Investigation	RepRel	
Relator Interrogation	RepRel	

Association Cycle

AssCyc Refactoring Options

The follwing options can be used to refactor the model.

Please choose which action to perform on the model:

- Create OCL invariant ENFORCING the instance level cycle
- [✓ Create OCL invariant FORBIDDING the instance level cycle]
- Create OCL derivation rule for one of the associations

Please, choose the association to be derived:

Anti-Pattern Modifications

Congratulations! You successfully verified the antipattern.

Added OCL rules:

```
context _'Criminal Investigation'
inv acyclic : self._'detective'._'interrogation'._'witness'._'criminal investigation'->excludes(self)
```

OK | Cancel

Moved: «kind» person, «role» detective, «role» lead detective, «relator» criminal investigation, «role» witness, «role» suspect, «role» senior detective, «role» junio detective, «relator» interrogation, «generalization» [unnamed], «mediation» h

Figure 19. Anti-Pattern detection and analysis capabilities incorporated in the OntoUML editor.

An anti-pattern analysis using the aforementioned wizard always ends up in one of the following conclusions ways: the occurrence characterizes a modeling problem fixable by a pre-defined refactoring plans or the identified structure is correct (thus the occurrence is a “false positive”). The tool presents the analysis results of the wizard and then the last step of our anti-pattern management strategy comes into play: the automatic elimination. In **Figure 19.5**, the tool is informing the user that it will include the following OCL invariant in the model to eliminate the undesired consequences of the anti-pattern occurrence.

In fact, **Figure 19** features the results of feeding the domain model of Criminal Investigation depicted in **Figure 2.** In this particular case, the detection algorithms identified nine occurrences of our ontological anti-patterns (**Figure 19.2**): two *AssCyc*, one *BinOver*, two *ImpAbs*, one *RelOver*, one *RelSpec* and two *RepRel*. In the following, we illustrate an example of *AssCyc* and on an example of *RelOver* in this model.

One identified *AssCyc* is a cycle composed by Criminal Investigation, Detective, Interrogation, Witness and, again, Detective (with the respective associations). This possible occurrence is shown in the dialog depicted in **Figure 19.3**, as the first item of the list. Using the OLED’s simulation component (discussed in Section 2), we obtain a visual representation of an instance of this model, as depicted in **Figure 20**. In this instance, investigation *Property2* has as witness *Object0,* who is questioned in interrogation *Property1* by detective *Object2*, who is member of investigation *Property3*, not investigation *Property2*. In other words, the model allows for a representation of a state of affairs in which an interrogation that is part of a criminal investigation is conducted by a detective that is not part of that investigation. Let us suppose that the creators of that model do not intend such a state of affairs. The modelers can then request the editor for an OCL solution that would proscribe instances with this detected unintended characteristic (**Figure 19.4**). In this case, the OCL constraint to be incorporated in the model (**Listing 7**) is the following:

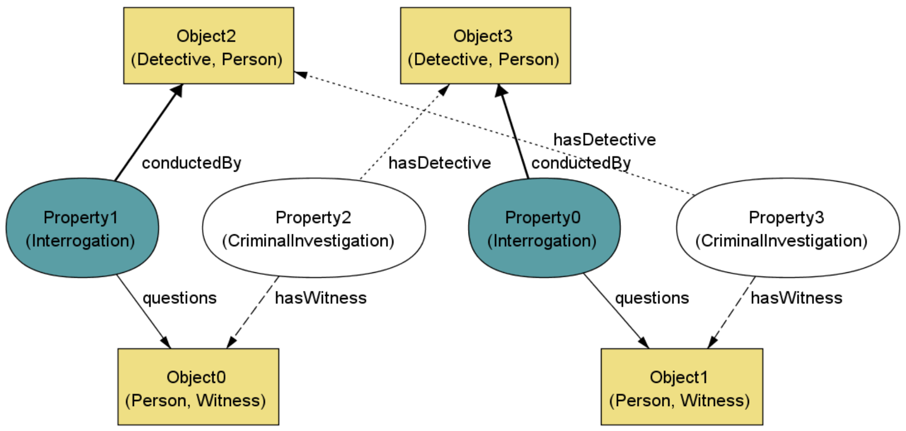
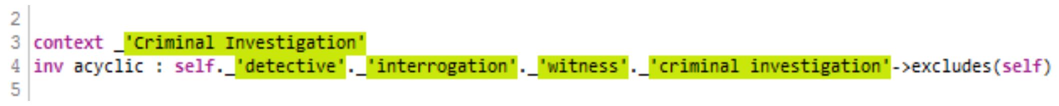


Figure 20. Possible interpretation of the *AssCyc* identified in the Criminal Investigation model.



```
context _'Criminal Investigation'
inv acyclic : self._'detective'._'interrogation'._'witness'._'criminal investigation'->excludes(self)
```

Listing 7. Auto-generated solution to forbid cycles at the instance level.

An example of an identified *RelOver* occurrence involves Criminal Investigation as a relator that mediates the Roles Detective, Lead Detective, Suspect and Witness. As explained in Section 4, there are three types of possibly unintended cases that can be allowed by an occurrence of this anti-pattern. First, all roles are exclusive in the scope of a particular relator, which means in this example that in each particular Criminal Investigation, the roles of Suspect, Witness, Detective and Lead Detective are necessarily all instantiated by different people. Second, it may be the case that only some of these roles are exclusive in the scope of a particular relator, for example, the Detective and the Suspect are exclusive, but not Detective and Witness, or Suspect and Witness. Finally, it may also be the case that some of the roles are disjoint (across different relators). For example, suppose the constraint that Detectives who participate in an ongoing Investigation cannot be considered a Suspect in another Investigation. Let us suppose that, as a first action to rectify the model, the modeler chooses to declare all roles as exclusive w.r.t. a given Investigation. The set of instances of the resulting model, hence, includes the one depicted in **Figure 21.** By inspecting such possible instance, the user can then realize that she perhaps overconstrained the model since, as a result of declaring all roles as exclusive, we have that the responsible for a given Investigation (i.e., the Lead Detectives) cannot be considered as a participant of that Investigation (i.e., one of its Detectives). The modeler can then once more rectify the model by choosing among a set of solutions offered by OLED. She might choose to declare the roles of Witness and Suspect disjoint w.r.t. a given Investigation (**Listing 8**), but also to declare that the roles of detective and suspect should be disjoint across different investigations, which the tool enforces by the creation of a generalization set.

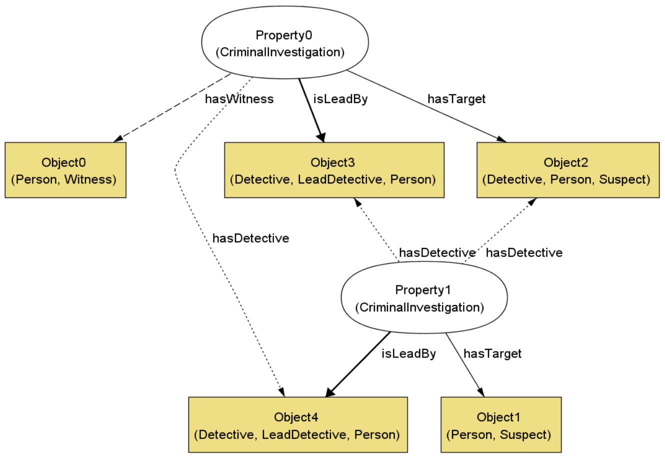
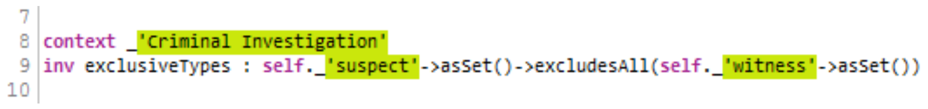


Figure 21. Exclusive view of the roles in a criminal investigation.



```
context _'Criminal Investigation'
inv exclusiveTypes : self._'suspect'->asSet()->excludesAll(self._'witness'->asSet())
```

Listing 8. Automatically generated OCL solutions to excluded unintended instances of RelOver.

6. Evaluating the Anti-Pattern Catalog

As previously discussed, in the second empirical study reported in section 2, given the sheer number of occurrences of anti-patterns across our repository, we were not able to check for each of them whether they were indeed a case of model fragments that entailed unintended consequences. In this section, we report a third empirical study in which we select one particular ontology in the repository to investigate a correlation between anti-pattern occurrence and whether they configured a case of an unintended model fragment. The goal of third study is to measure two things: the likelihood that an anti-pattern occurrence characterizes a domain misrepresentation; how often users select the refactoring plans (proposed in section 4 and made available in the tool presented in section 5) to solve an occurrence that in fact characterizes a mistake. Through the analysis of these two variables, we assess the usefulness of an anti-pattern characterization.

A case for this third empirical study, we have selected the MGIC reference model. The rationale for this decision includes the following observations:

- The ontology is the biggest in the repository;
- It contains occurrences of all anti-pattern types;
- Ten modelers participated in its development, throughout three years;
- It is the product of an industrial project with the Brazilian government; and
- Most importantly, the modelers accepted to participate in the analysis because it was of their own interest as they needed to validate their reference model.

6.1 Methods and Tools

Eight modelers participated in this empirical study. We assigned sub-ontologies to each of them, taking into account their knowledge of the domain. To guarantee that the modelers would have enough knowledge to analyze the anti-pattern occurrences, we aimed at assigning to a modeler those models in which development they had participated. We also encouraged modelers to interact with each other during the case study to share experiences. Modelers conducted the anti-pattern detection exclusively through the OntoUML tool. They also analyzed the occurrences exclusively using the model wizards that we have implemented (see Section 5).

We intentionally did not provide participants with any anti-pattern training, not regarding their structure, justification or predicted solutions. We also did not specify any order in which the participants should analyze the anti-patterns. We made these decisions because our goal, since the beginning of this research, was to develop a tool that did not require formal training. In fact, that is one of the reasons that we implemented a wizard to guide modelers throughout the analysis of each anti-pattern.

We highlight that, throughout this empirical study, the anti-pattern management tool was improved. We established an interactive process: the participants used the tool and whenever they identified bugs or improvements opportunities, they contacted us so we could discuss and improve the wizard or the refactoring options. We were available to answer all sorts of question to the participants, either regarding anti-pattern definition, refactoring plans or even ontological notions required to understand the anti-patterns.

We asked each participants to manually analyze each anti-pattern occurrence identified within his assigned subdomain and to register his conclusion according to the following template:

- **Anti-Pattern Type:** an acronym to identify the type of anti-pattern
- **Description:** a textual description automatically generated by the anti-pattern tool. It identifies the classes and associations relevant for understanding the anti-pattern. Moreover, it is useful to reproduce each occurrence.
- **Decision:** a binary field that captures the ultimate decision regarding an anti-pattern occurrence. The field can be set as "Error" or "Correct". Participants used the former if the occurrence analysis lead to some modifications in the model, predicted by our anti-patterns or not. We intentionally did not provide a "Don't Know / Don't Understand" option because, in those cases, we instructed the modelers to interact with us, if the doubt regarded anti-pattern definition, or to interact with each other, to arrive at a decision question.
- **Action:** describes the action participants adopted to refactor the model – we instructed participants to input information in this field only if they identified an error.
- **Predicted:** three values can be assigned to this field: "Yes", "No" and "Partially". It refers to whether anti-pattern was completely able, partially able or unable to predict the refactoring actions.
- **Comment:** a field that participants could freely fill (e.g. doubts, intuitions, observations, and so on).

6.2 The MGIC Ontology

Before we present the study results, we briefly describe the MGIC ontology, providing an overview of its development context, the domain it formalizes and its structural information (number of classes, relations, stereotypes, etc.).

The project entitled **Model for Information and Knowledge Management (MGIC)** is a product of a partnership between the Brazilian Ground Transportation Regulatory Agency (ANTT) and the Brazilian federal universities of Espírito Santo, Fluminense and Rio de Janeiro (UFES, UFF and UFRJ respectively). The project was conceived to improve decision making within the Agency, by means of an information and knowledge management model [22]. The adopted methodology proposed the creation of five types of models: information flow, business requirements and assets, knowledge and competence, and an ontology-based reference conceptual models. The main role of the ontology-based models was to provide structure and semantics to the information handled by the agency, to serve as a reference model to allow semantic interoperability between the systems controlled by the agency [22]. The MGIC ontology also intended to serve as a guide to the agency's databases triplication and publication. Information transparency and publication is a law-imposed obligation for all Brazilian governmental organizations since the sanction of the bill entitled *Free and Open Access to Information Act* [38]. The design of the MGIC ontology was carried out over a period of 3 years. Throughout that time, 10 modelers were involved, who collaborated with nearly 40 domain experts in order to define the scope and capture the conceptualization shared within the agency. A team of ontologists visited the 11 main departments of the agency. In each department, they interviewed experts appointed by the departments' management. The model describes the subdomains relevant for ground transportation regulation including the following:

- **Cargo transportation:** concepts related to cargo transportation by truck, train, pipelines or multimodal (a combination of different transportation means). It describes the differences between interstate and international cargo transportations, and transportation of hazardous products.
- **Passenger transportation:** describes concepts related to interstate and international regular and eventual passenger transportation on both highways and railroads.
- **Infrastructure concession:** describes the process of concession and controlling of highway and railroads infrastructure to private companies.
- **Legislation:** includes concepts regarding the legal process for regulating the transportation sector.

From a structural perspective, the MGIC Ontology is massive, particularly if compared to other conceptual models. It contains 3800 classes, 1918 associations, 3616 generalizations, 698 generalization sets, 71 data types, 865 attributes and 149 constraints, all distributed over 291 packages. The ontology also has occurrences of every single language construct defined in OntoUML: all eleven class' stereotypes and all nine association's stereotypes.

In particular, the Role construct is the most frequently used class stereotype: 1066 occurrences or 28.1% of all classes, whilst the most used association type was *mediation*: 1103 out of 1918 (or 57.5%).

6.3 Results

The modelers managed to analyze all 879 anti-pattern occurrences identified in the MGIC ontology. We summarize the results in **Table 12**. The column identified as "#Occ.", stands for the number of analyzed occurrences of a given anti-pattern type, whilst the one labeled as "#Error", refers to the number of occurrences considered as modeling errors by the participants. The columns "#Refac.", "#Partial" and "#Custom" stand for the sum of occurrences the participants fixed using: exclusively refactoring plans, some refactoring plans and some custom solutions and exclusively custom solutions, respectively. We measure the probability of an anti-pattern occurrence to characterize a mistake (entitled anti-pattern *problem rate)* by dividing the number of times it characterizes a mistake (#Error) by the number of times it occurs (#Occ.). We measured the capability of an anti-pattern to anticipate the solution for an erroneous occurrence (entitled anti-pattern *predictability*) by dividing the number of erroneous occurrence in which only refactoring plans were used (#Refac.) by the number of erroneous occurrences.

Table 12. Summary of the Anti-Patterns accuracy results.

Anti-Pattern	#Occ.	#Error	#Error / #Occ.	#Refac.	#Refac. /#Error	#Partial	#Partial / #Error	#Custom	#Custom / #Error
RelSpec	315	279	88.6%	271	97.1%	1	0.4%	7	2.5%
RepRel	221	57	25.8%	48	84.2%	4	7.0%	5	8.8%
RelOver	124	70	56.5%	54	77.1%	1	1.4%	15	21.4%
BinOver	74	31	41.9%	23	74.2%	0	0.0%	8	25.8%
AssCyc	20	14	70.0%	10	71.4%	0	0.0%	4	28.6%
ImpAbs	125	11	8.8%	3	27.3%	0	0.0%	8	72.7%
Total	879	462	52.56%	409	88.53%	6	1.46%	47	10.17%

As one can observe, we found some variability regarding anti-pattern *problem rate*. For instance, while for *RelSpec*, 88.6% of their occurrences actually configured a case of an unintended model fragment, this percentage was only of 8.8% for the case of *ImpAbs*. In general, 4 out of 6 of our anti-patterns configured cases of unintended consequences in more than 50% of the cases (bringing the total problem rate to 52.56%). These numbers are a strong indication that the structures identified by the anti-patterns are indeed error-prone.

Anti-pattern *predictability* refers to the capacity of predicting appropriate refactoring solutions. We measure that by dividing the number of occurrences in which modelers exclusively adopted standard solutions by the number of occurrences that they considered as mistakes. In this study, the modelers exclusively adopted pre-defined solutions 409 times in the 462 occurrences considered as errors. This represents a percentage of 88.53% of the time, in comparison to 1.46% of partial solutions and 10.17% of exclusively custom ones. We observed a much lower variability on the effectiveness of the proposed solutions for each of the patterns: for 5 out of 6 patterns, the solutions were adopted as proposed by the tool in more than 70%. In the case of *RelSpec*, the solutions were accepted as proposed in 97.1% of the cases.

If we inspect individual anti-patterns, we can also observe certain variability regarding the adopted solutions. For instance, the BinOver anti-pattern (characterized by an association between overlapping types) occurred 74 times. From those, 31 were actual mistakes (41.9%). Considering only the erroneous cases, the modelers adopted one of our suggested solutions 23 times (74.2%). The option to enforce one or more binary properties was selected 16 times (70%), while the alternative to enforce *disjointness* between the related types was selected seven times (30%). In this study, the modelers did not opted to change the stereotype of the relation for any occurrence. If we inspect the types of enforced binary properties, we see that anti-reflexivity and anti-symmetry were the most common ones, being selected 15 and 14 times respectively. The acyclic constraint comes next being selected 9 times. The need to specify transitive, reflexive or symmetric relations was only encountered one, two and one time respectively. This is an indication that the need for explicitly specifying binary meta-properties exists and that we should try to pro-actively incentive modelers to specify such constraints. We also managed to identify the reasons that lead the modelers not to consider a BinOver occurrence to be a mistake. From the 43 correct cases, in 19 times (44.2%) they considered the relation under analysis as derived and, as such, the binary meta-properties were consequence of the embedded derivation rules. In another 11 times (25.6%), the desired binary meta-properties were determined by the relation stereotype choice. Furthermore, in seven cases (16.3%), the participants fixed the problem unintentionally, through the adoption of a solution to another anti-pattern. Lastly, for the remainder 6 cases, participants did not provide further justification.

To provide a conclusion on anti-pattern evaluation, we cross *frequency* (measured in **Table 3**), *problem-rate* and *predictability* information on **Table 13**. For the frequency column, we consider the percentage of models an anti-pattern was encountered given the models in which they could occur (i.e., models with at least one instance of the anti-pattern's main element type). For the problem rate and predictability columns, we considered the results of the MGIC case study. In the former, we considered the rate between occurrences that characterized errors per total number of occurrences, and in the latter, the percentage of erroneous occurrences that one could fix using exclusively pre-defined solutions. Furthermore, instead of the actual percentages, we adopted a discrete scale with values specified as follows: Very High (80-100%), High (60-80%), Medium (40-60%), Low (20-40%) and Very Low (0-20%).

The higher all these three values are for an anti-pattern, the more useful it is for validation. Anti-patterns that always occur, with a high possibility of characterizing a mistake and being able to predict most of the refactoring necessities are more likely to be useful during conceptual model validation. Examples of such anti-patterns that fit the profile are *AssCyc* and *RelSpec*. Less useful anti-patterns, on the other hand, are not the scarcely identified ones, but the ones that we frequently find but rarely are the source of domain misrepresentations. In fact, they require a lot of effort to analyze and little gain in ontology quality. *ImpAbs* is an example of anti-pattern that needs refinement. Our analysis of the phenomenon is that *ImpAbs'* generic structure, which encompasses all association and class stereotypes, dramatically increases the *frequency*. This overly generic structure is also the cause of the lower *predictability,* since it broadens the types of problem.

Table 13. Summary of the evaluation results from both studies.

Anti-Pattern	Frequency	Problem Rate	Predictability
AssCyc	Very High	High	High
BinOver	Medium	Medium	High
ImpAbs	High	Very Low	Low
RelOver	Low	Medium	High
RelSpec	Medium	Very High	Very High
RepRel	High	Low	Very High

Overall, the combination of this third empirical study, presented in this section, combined with the frequency evaluation performed in the second study (presented in Section 3) significantly improved our confidence that anti-patterns are a very useful tool in conceptual model validation and that our catalog and tool support are evidences of that.

7. Final Considerations

This paper makes a contribution to the theory and practice of ontology-driven conceptual modeling by: (i) presenting a number of empirically elicited Ontological Anti-patterns that were identified as recurrent in a benchmark of conceptual models; (ii) precisely characterizing these anti-patterns as well as proposing a number of model refactoring plans that can be used to remove from the models the unintended consequences cause by the occurrence of each of these anti-patterns; (iii) a computational environment that automates the process of supporting detection of occurrences of these anti-patterns, exploration of their consequence in individual models, formal rectification via the inclusion of pre-defined formal constraints (implementing the aforementioned refactoring plans). This computational environment is available in https://code.google.com/p/ontouml-lightweight-editor/; (iv) an empirical study that elucidates both the harmfulness of each of these anti-patterns (i.e., how likely is that they in fact introduce unintended consequences) as well as the effectiveness of the proposed systematic and automated solutions to rectify their harmful occurrences.

During our empirical studies, we have noticed a change in the modelers' behavior that is worth reporting here. We noticed that by using the tool, the modelers get so familiar with the anti-patterns that they start pro-actively identifying their occurrences while developing new models. In other words, by being aware of the possible solutions that can used to avoid these anti-patters, the modelers seem to develop modeling skills that systematically prevent these anti-pattern's occurrences. We would like to systematically explore this observation in a future empirical study.

The next step that we envisage for this research is to develop more advanced pro-active ways to prevent the occurrence of anti-patterns in conceptual models. One way to do that is to propose the inclusion of new modeling configurations for OntoUML constructs. For instance, as explained in [10], OntoUML is a pattern language and, as such, its modeling constructs are clusters of constructs amounting to an *Ontological Design Pattern.* An example is the *Relator-Material Relation (RMR) Pattern* briefly discussed in section 2. By understanding the causes behind the occurrences of anti-patterns such as RelOver or RepRel, we can provide new configuration options for RMR. For instance, we can include a modeling option for defining the possible constraints between the types mediated by a relator (e.g., declaring them mutually exclusive) or a meta-attribute for specifying the number of times the same elements can be connected by different instances of the same relator type.

Of course, as always, one must analyze the trade-off between language expressivity and complexity. In fact, this trade off should also be analyzed when developing educational material for ontology-driven conceptual modeling. Regarding this, we believe that empirical studies such as the one reported in section 6 can be of paramount value. On one hand, it inform us which anti-patterns tend to frequently occur and which ones tend to frequently introduce unintended consequences in the models. Moreover, from a research perspective, it also gives evidence to which anti-patterns we have already developed effective solutions.

Since Koenig's original proposal [11], the concept of anti-pattern has been applied in a variety of fields other than the original field of software design. To the extent of our knowledge, however, there is no other application of anti-patterns in ontology-driven conceptual modeling. Nonetheless, our approach is in line with authors both in the conceptual modeling and ontology engineering/semantic web literature. Representative examples of works in the latter area are [39] and [40], which discuss methods of detecting anti-patterns in OWL specifications via SPARQL queries. Although sharing the same general objective, our approach differs from these works in a number of important ways. Firstly, our approach is based on a much richer modeling language from the ontological point of view. As a consequence, the anti-patterns addressed by our approach are able to address more subtle ontological conditions such as, for example, the ones involving modality, identity principles as well as a richer ontology of material relations. Secondly, different from these approaches, our method does not aim at detecting general cases involving typical logical misunderstandings. In contrast, it focuses exactly on those cases that cannot be casted as modeling (grammatical) errors by the process of formal verification, and aims at identifying recurrent potential deviations between the sets of valid and intended model instances. Thirdly, for instance in [40], the identified anti-patterns are cases believed to be caused by the lack of modeling experience [40]. Here, as shown by our empirical studies, these anti-patterns are recurrent even in models produced by experience researchers. In fact, in pace with [10], we believe that the repeated occurrence of these anti-patterns is an intrinsic feature of the disparity between the increasing complexity of our reference conceptual models and our limited cognitive capacities for dealing with that. Finally, in contrast with these approaches, besides automatic anti-pattern detection, our approach presents a computational environment for model analysis (via visual simulation) and systematic conceptual model rectification.

Acknowledgements. This research was partially supported by the Lucretius ERC Advanced Grant # 267856. The authors would like to thank the ANTT employees and members of the MGIC project that collaborated with this research.

References

1. Weber, R., Ontological Foundations of Information Systems, Coopers & Lybrand, Melbourne, 1997.
2. Dijkstra, E.W., The Humble Programmer, Communications of the ACM, 15:10, Oct. 1972.
3. Guarino, N., Musen, M., Applied Ontology: Focusing on Content, Applied Ontology, Vol. 1, pp. 1-5, IOS Press, 2005.
4. Guizzardi, G., Herre, H., Wagner G., On the General Ontological Foundations of Conceptual Modeling, 21st International Conf. on Conceptual Modeling (ER 2002), Tampere, 2002.
5. Wand, Y., Weber, R., An Ontological Model of an Information System. IEEE Trans. Software Eng. 16(11): 1282-1292, 1990.
6. Recker, J., Rosemann, M., Green, P., Indulska, M., Do Ontological Deficiencies in Modeling Grammars Matter?, MISQ Quaterly, Vol. 35, n. 1, 2011.
7. Guizzardi, G.: Ontological foundations for structural conceptual models. Centre for Telematics and Information Technology, University of Twente, The Netherlands, (2005).
8. Benevides, A.B., Guizzardi, G.: A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML, 11th International Conf. on Enterprise Information Systems (ICEIS), Milan, 2009.
9. Benevides, A.B. et al.: Validating Modal Aspects of OntoUML Conceptual Models Using Automatically Generated Visual World Structures. Journal of Universal Computer Science. 16, 2904–2933, 2010.
10. Guizzardi, G., Ontological Patterns, Anti-Patterns and Pattern Languages for Next-Generation Conceptual Modeling, invited companion paper to the Keynote Speech delivered at the 33rd International Conference on Conceptual Modeling (ER 2014), Atlanta, USA.
11. Koenig, A.: Patterns and Anti-Patterns. J. of Object-Oriented Programming. 8 (1995).

12. Guizzardi, G., On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models, Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV, IOS Press, Amsterdam, 2007.
13. Guizzardi, G., Wagner, G. Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages In: Theory and Application of Ontologies ed.Berlim: Springer-Verlag, 2010.
14. Guizzardi, G., Sales, T.P., Detection, Simulation and Elimination of Semantic Anti-Patterns in Ontology-Driven Conceptual Models. Proc. of 33rd International Conf. on Conceptual Modeling (ER 2014), Atlanta.
15. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge, Massachusetts (2012).
16. Costal, D., Goméz, C., Guizzardi, G., Formal Semantics and Ontological Analysis for Understanding Subsetting, Specialization and Redefinition of Associations in UML, 30th International Conference on Conceptual Modeling (ER 2011), Brussels, Belgium, 2011.
17. Sales, T.P., Barcelos, P.P.F., Guizzardi, G.: Identification of Semantic Anti-Patterns in Ontology-Driven Conceptual Modeling via Visual Simulation. 4th International Workshop on Ontology-Driven Information Systems (ODISE), Graz, Austria (2012).
18. Barcelos, P.P.F.; Guizzardi, G.; Garcia, A.S., Monteiro, M.E., Ontological Evaluation of the ITU-T Recommendation G.805, 18th International Conference on Telecommunications (ICT 2011), IEEE Press, Cyprus
19. Albuquerque, A., Developing a Domain Ontology for Biodiversity (in Portuguese), Master Dissertation, Federal University of Amazonas, Manaus, Brazil 2011.
20. Gonçalves, B.N.; Guizzardi, G.; Pereira Filho, J.G., Using an ECG reference ontology for semantic interoperability of ECG data, Journal of Biomedical Informatics, Special Issue on Ontologies for Clinical and Translational Research,Editors: Barry Smith, Werner Ceusters and Richard H. Scheuermann, Elsevier, 2011.
21. Brazilian Ministry of Planning and Management, A Preliminary Conceptual Model for ther Brazilian Governmental Organizational Structures, available at the Persistent Ontology Repository of the National Program on Government Interoperability (E-PING), online: http://vocab.e.gov.br/.
22. Bastos, C.A.M., Rezende, L., Caldas, M., Garcia, A.S., Mecena Filho, S., Sanchez, M.L., Castro Junior, J.L.P., Building up a Model for Management Information and Knowledge : The Case-Study for a Brazilian Regulatory Agency, in International Workshop on Software Knowledge (SKY), 2011.
23. Calhau, R., Falbo, R., A Configuration Management Task Ontology for Semantic Integration, in ACM Symposium on Applied Computing (SAC), 2012, pp. 348–353.
24. Nardi, J.C., Falbo, R.A., Almeida, J.P., Guizzardi, G., Pires, L.F., van Sinderen, M., Guarino, N., Towards a Commitment-Based Reference Ontology for Services, in IEEE Enterprise Distributed Object Computing Conference (EDOC), 2013, pp. 175–184.
25. Ferrandis, A.M.M., Lopez, O.P., Guizzardi, G., Applying the Principles of an Ontology-Based Approach to a Conceptual Schema of Human Genome, in International Conference on Conceptual Modeling (ER 2013), pp. 471–478.
26. Ferreira, M.I.G.B., Cordeiro, K.F., Oliveira, J., Campos, M.L.M., OntoEmerge: The Construction of a Core Ontology for the Emergency Domain based on a Foundational Ontology (in Portuguese) in 4th Brazilian Seminar on Ontological Research (ONTOBRAS 2010), Gramado, Brazil.
27. Ferreira, M.I.G.B., An Emergency Ontology supporting the Generation of Vairability Solutions (in Portuguese), Master Dissertation, Federal University of Rio de Janeiro, Brazil, 2013.
28. Pereira, D.C., Almeida, J.P., Representing Organizational Structures in an Enterprise Architecture Language, in Workshop on Formal Ontologies meet Industry (FOMI), 2014, pp. 7–15.
29. Pereira, D.C., Representing Organizational Structures in Enterprise Architecture: an Ontology-based Approach, Ontology and Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo, Vitória, Brazil, 2015.
30. Cruz, S.M.S, Campos, M.L.M., Mattoso, M., A Foundational Ontology to Support Scientific Experiments, in 6th Brazilian Seminar on Ontological Research (ONTOBRAS), 2010, 2012, pp. 144–155.
31. Silva, H.C., Castro, R.C.C., Gomes, M.J.N., Garcia, A.S., Well-Founded IT Architecture Ontology: An Approach from a Service Continuity Perspective, in 4th International Conference on Networked Digital Technologies (NDT'12), 2012, vol. 294, pp. 136–150.
32. Castro, R.C.C., Silva, H.C., Garcia, A.S., Gomes, M.J.N., Mapping of vulnerabilities in the public cloud with the use of foundational ontology: A perspective for service IaaS, in 7th International Conference on Digital Information Management (ICDIM'12), 2012, pp. 245–252.
33. Barcelos, P.P.F., Guizzardi, R.S.S., Garcia, A.S., An Ontology Reference Model for Normative Acts, in 7th Brazilian Seminar on Ontological Research (ONTOBRAS) 2013, pp. 35–46.
34. Santos, V.A., Reginato, C.C., Komati, K.S., Monteiro, M.E., Conceptual Mapping Between the OpenFlow Protocal and the ITU-T G.805 Recommendation (in portuguese), in Computer on the Beach, 2013, pp. 340–342.
35. Machado, B.N., Semantic Documentation in Requirements Engineering, Federal University of Espírito Santo, Technical Report, Vitória, Brazil, 2012.
36. Schimidt, G.; Strohlein, T. Relations and Graphs: Discrete Mathematics for Computer Scientists. 1st. ed. Berlin, Germany: Springer, 1993. p. 301
37. Brazilian Free and Open Information Acess Act (in portuguese: Lei de Acesso à Informação). Law no 12.527, November 18th, 2011.
38. Vrandečić, D.: Ontology Validation, PhD Thesis, University of Karlsruhe (2010).
39. Roussey, C. et al.: SPARQL-DL queries for Antipattern Detection. Workshop on Ontology Patterns. CEUR-WS.org, Boston, USA (2012).