On investigating Metamodel Inaccurate Structures

ABSTRACT

Metamodeling allows to capture domain knowledge through the definition of its structure (concepts and relations between them) and its constraints (logical expressions) often written in OCL. The OCL constraints added to a metamodel are of two types: 1) domain-related constraints: they differ from one domain to another and are expressed based on the knowledge of experts; and 2) those that are added to the majority of metamodels to precise some inaccurate structures that may cause problems when instantiating models. We call these structures Metamodel Inaccurate Structures (MIS). In this paper, we performed an empirical study in order to point out the metamodel inaccurate structures. As a first step, a study is conducted on a set of OCL constraints taken from the UML metamodel to investigate the relation between structure and constraints. Then, to confirm our findings, we realised a quantitative analysis in order to count the occurrences of constraints that complete inaccurate structures. We believe that our results can help designers in the quest of creating or refactoring metamodels and specifying constraints that precisely capture domain knowledge to ensure consistency of the derived artifacts.

KEYWORDS

Metamodeling, OCL, MOF, Well-Formedness Rules, Metamodel Inaccurate Structure, MDE, Empirical Study

ACM Reference Format:
. 2020. On investigating Metamodel Inaccurate Structures. In *Proceedings of ACM SAC Conference (SAC'20).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

In Model-Driven Engineering (MDE), a metamodel represents the abstract syntax of a Domain-Specific Modeling Language (DSML) [26]. A metamodel structure captures core domain concepts, and relationships between them. Considering the difficulty or the impossibility to express some information in a diagrammatic way, textual constraints, also called Well-Formedness Rules (WFR), are specified on metamodels to restrict the scope of some defined concepts. This ensures the semantic correctness of the generated model instances. Consequently, one can benefit from the full power of model-driven engineering only if the metamodel is precise enough to adequately describe both the syntactical and semantic parts of the intended domain.

SAC'20, March 30-April 3, 2020, Brno, Czech Republic
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/xx.xxx/xxx_x

Currently, most metamodels present in repositories (such as OMG [16]) include only a description of the structural part [15]. WFRs are rarely included, and sometimes the set of WFRs is roughly specified and hence does not prevent all semantic inaccuracies. This is mainly due to the fact that WFR elicitation is performed manually, which is a time-consuming and error-prone task. Simplifying this tough procedure was the objective of many works. Several approaches have been explored, in particular using metamodel structure and a set of correct and incorrect models to generate OCL constraints([11, 15]). Other approaches have directly targeted the OCL language in order to identify constraint patterns allowing to specify most of OCL constraints([4, 6, 7, 18, 19, 34]). One can notice that some OCL constraints are recurrent regardless the represented domain. We believe that some metamodel structures are at the root of the need of these constraints.

Thus, this work aims to investigate the role of some metamodel structures in the existence of certain OCL constraints. Our objective is to point out the structures that are generally constrained. For this, we studied metamodels that already include OCL constraints. We analyse manually each constraint with its metamodel targeted structure to identify structures that cause the inaccuracy, and which led to the definition of the constraint. It is important to emphasize that depending on the targeted domain, the presence of these structures does not automatically imply inaccuracies. However, it indicates a potential problem that should be inspected by the metamodel designer to check whether it is a real one, hence the suggested name "Metamodel Inaccurate Structure" (MIS). We also believe that the Unified Modeling Language (UML) metamodel is the most suitable metamodel that we can rely on in our study to identify these MISs.

The rest of the paper is organized as follows. In Section 2, an illustrative example of Metamodel Inaccurate Structures is provided. Section 3 presents the experimental design. In Section 4, we report the results of our analysis. Section 5 depicts the threats to validity of our findings. Before concluding the paper in Sections 7, we describe some related work in Section 6.

2 ILLUSTRATIVE EXAMPLE

As described in [15], a metamodel is defined as the composition of: i) a domain structure, which encompasses the core concepts and attributes that define the domain as well as relationships between these concepts. ii) well-formedness rules, which are additional constraints applied on concepts. These restrict the way the structural elements can be instantiated and assembled to form a valid model with respect to the domain semantic. In our study, we used metamodels that are formalised with the Meta-Object Facility (MOF), and well-formedness rules with the Object Constraint Language (OCL).

To precisely draw the distinction between the two kind of constraints mentioned above, we use an example (see Fig. 1) taken from the UML metamodel (Activity Diagrams).

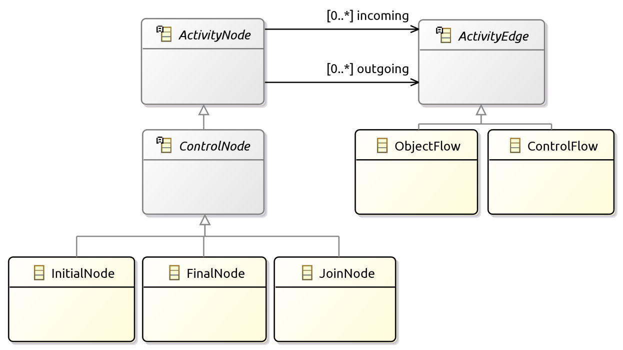

Figure 1: UML Activity diagram structure

Generalization (Inheritance) allows to group common elements (associations, attributes, and operations) from different classes sharing some abstract definitions, and merge them into one class. Consequently, classes in need of these properties inherit from another class (holding these properties) without any local redefinition. Generalization allows to simplify metamodel structure and avoid redundancies. However, merging properties from different classes into one class often leads to some loss of precision. This is because while merging many properties into one, the widest multiplicity range is taken to include all the multiplicities. For instance, in Fig. 1(taken from the UML metamodel [17]), the three concrete subclasses *InitialNode, FinalNode,* and *JoinNode* are indirectly associated to *ActivityEdge* class through the associations *incoming* and *outgoing* (both having a multiplicity of [0..*]) that defined in the super-class *ActivityNode*. With respect to UML semantics, to get a correct UML activity diagram instance, the following invariant must be respected:

(1) A *JoinNode* must have only one outgoing *ActivityEdge*.
(2) An *InitialNode* in an activity diagram can not have incoming *ActivityEdges*.
(3) A *FinalNode* does not have outgoings.

Given the metamodel structure, the listed rules are not explicit. To avoid the loss of information caused by the generalization, three solutions are possible. The first one consists in refactoring the metamodel to avoid generalization when it is possible. This solution will have an impact on the metamodel's complexity and maintainability aspects. The second solution consists in specializing the association in subclasses. In fact, this solution is the best alternative from a model perspective, although it makes the metamodel more complicated. The third solution, and the most recommended, is to refine the semantics of the metamodel through OCL constraints, without refactoring the metamodel structure. For instance, to enrich the UML metamodel semantics the above constraints were expressed in OCL as follows:

WFR1: A *JoinNode* has only one outgoing *ActivityEdge*.

$$\textbf{context}\ \text{JoinNode}\ \textbf{inv}:\\ \text{outgoing} -> \text{size}() = 1$$

WFR2: An *InitialNode* has no incoming *ActivityEdges*.

$$\textbf{context}\ \text{InitialNode}\ \textbf{inv}:\\ \text{incoming} -> \text{isEmpty}()$$

WFR3: A *FinalNode* has no outgoing *ActivityEdges*.

$$\textbf{context}\ \text{FinalNode}\ \textbf{inv}:\\ \text{outgoing} -> \text{isEmpty}()$$

Even if we assume that OCL constraint specification process can be carried out by metamodel designer having domain knowledge, the need of specifying OCL constraints in this kind of situations is structurally discernible. Indeed, by looking only at the structure, without having domain-knowledge, we can wonder whether or not the domain semantics is fully expressed diagrammatically. In our example, the multiplicity of associations *incoming* and *outgoing* attracts the attention. This leads us to wonder if all subclasses of the class *ActivityNode* really need such a large level of multiplicity. This is why we refer to this kind of structures as "metamodel inaccurate structures".

Conversely, the existence of certain constraints can not be guessed just by looking at the structure of the metamodel without knowing precisely its semantics. For example, consider the following constraint:

WFR4: If one of the incoming *ActivityEdges* of a *JoinNode* is an *ObjectFlow*, then its outgoing *ActivityEdge* must be an *ObjectFlow*. Otherwise its outgoing *ActivityEdge* must be a *ControlFlow*.

Clearly, this constraint reflects a domain-specific semantics and can in no way be suspected by the simple analysis of the structure of the metamodel. So, we can not consider the structure related to this constraint as a metamodel inaccurate structure.

3 EXPERIMENTAL DESIGN

We believe that the best way to point out metamodel inaccurate structures is to analyse already existing metamodels having OCL constraints. By studying where the constraints have been added, and why these constraints are added, we will be able to conclude if we face a MIS or not.

3.1 Research Questions

The study aims at addressing the following two research questions (RQs):

- ***RQ1:*** *What are the structures that are often completed with OCL constraints to complete their semantics?* This research question aims at identify structures that are often completed with OCL constraints. Specifically, we want to characterize metamodel fragments that are not enough precise to fully capture all semantic details, and that can lead to interpretation ambiguities, i.e. from these structures, it is possible to create artifacts that do respect metamodel structure but do not represent domain.
  To do so, we have established the following process:
  (1) Starting from a metamodel with well defined OCL constraints, for each OCL constraint identify its targeted metamodel fragment S_i.
  (2) Manually analyse each identified metamodel fragment S_i, by four metamodeling experts, in order to conclude if the structure by itself suggests the need for an OCL constraint to precise the semantics of the domain.

(3) Generalize the definition of the metamodel fragment in order to obtain a characterization of the corresponding MEBS.

- ***RQ2:*** *What is the proportion of the constraints that are applied to complete the identified metamodel inaccurate structures?* This question aims at investigating the frequency of OCL constraints related to each MIS and whether they are present in most of metamodels, or they are just related to the domain of one of the studied metamodels. To do so, we classified the OCL constraints by MIS, and then quantified each set of constraints to get the number of OCL constraints related to each MIS.

3.2 Data

Since we rely on existing metamodels to characterize our metamodel inaccurate structures set, metamodel choice is extremely important for the quality of the findings, especially the set of OCL constraints that need to be as complete as possible. Nonetheless, finding metamodels refined with OCL constraints is tough. Indeed, most metamodels present in repositories ([31] for example) do not include well-formedness rules, and in some cases, constraints are expressed in natural language.

Source	metamodel	Nb of Constraints
	UML 2.4	450
OMG	SysML 1.5	21
	ODM 1.1	19
	CWM 1.1	96
	Diagram Definition	16
Research Project	SAD3	8
	CPFSTool	27
	ER2RE	59
	RBAC	34
Industry	SAM	82

Table 1: metamodels for BS identification

At the end, we collected five metamodels from OMG [16] and five from ReMoDD repository [1]. Table 1 details the list of metamodels, with the number of OCL constraints for each.

4 ANALYSIS OF THE RESULTS

This section reports our analysis of the results that we achieved is the study. This aims at answering the two research questions formulated in Section 2.

4.1 What are the structures that are often completed with OCL constraints to complete their semantics?

In the following, we present each identified MIS in a separate subsection. The title of the subsection is the name of the MIS. As already said, a fragment of the metamodel is suspected of being an MIS when experts are brought to ask a question about it. Thus, for each identified MIS we give the formulated question.

4.1.1 Attribute Value Restriction. In MOF, there is no means to specify the accurate values that an attribute can take in diagrammatic way. As a consequence, the attribute may take all the possible values depending on its type (i.e. Integer can take a value from ℕ). However, some values are not correct with respect to domain semantics. The only way to restrict the attribute value is to write OCL constraints that specify the correct values range according to the domain, and hence exclude the incorrect values to avoid semantic inconsistencies. This MIS is related to the question: does the default attribute definition domain match the definition domain dictated by the semantics of the domain?

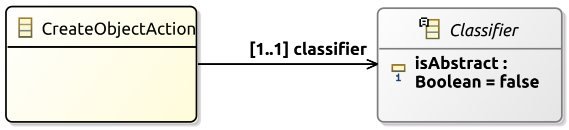

Figure 2: Attribute Value Restriction

As illustrated in Figure 2, a *Classifier* contains an attribute *isAbstract* that indicates if the Classifier is abstract or no. A subset of Classifiers related to *CreateObjectAction* should be abstract. The following OCL constraint is then specified.
WFR: The classifier cannot be abstract [17].

context CreateObjectAction **inv** :
not classifier.isAbstract

4.1.2 Enumeration Literals Restriction. An Enumeration is a DataType whose values are enumerated in the model as literals [17]. Thus, an Enumeration attribute can take as value one of the literals. If we can find several attributes with the same enumeration type, it is possible that the set of literals defined in the Enumeration is the union of the literals that each attribute may take. In this case, it is necessary to specify the subset of literals that each attribute may take in its class with an OCL constraint. While finding multiple attributes having the same enumeration type, one can ask the following question: do the Enumeration attributes accept all the listed literals in the Enumeration?

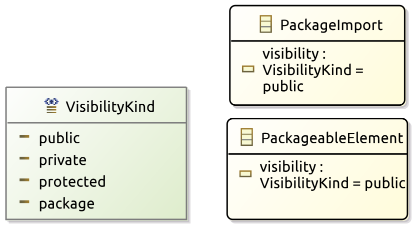

Figure 3: Enumeration Literals Restriction

For instance, in Figure 3, both *PackageableElement* and *PackageImport* classes contain a *VisibilityKind* attribute. Thus, each of the two attributes may take as value one of the literals of the *visibilityKind* enumeration (public, private, protected, package). However, following the UML semantic, *PackageImport* visibility can not be protected nor package. *WFR*: The visibility of a PackageImport is either public or private [17].

context PackageImport **inv** :
visibility = VisibilityKind::public or visibility = VisibilityKind::private

4.1.3 *Inherited Optional Attribute.*

We did notice that often on big metamodels, some attributes in superclass are defined as optional (having [0..1] bounds). Indeed, in some subclasses the inherited attribute is mandatory and must be always specified, and in other subclasses the same attribute should not exist. In this case, an OCL constraint should be written in order to specify the accurate attribute bounds. For this MIS, one might ask the following question: is there a subclass where this attribute is mandatory and must always be specified? or ambiguous, and must be excluded?

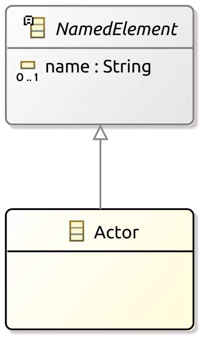

Figure 4: Inherited Optional Attribute

As illustrated in Figure 4, the optional attribute *name* defined in NamedElement is mandatory in the sub-class *Actor* in UseCase diagram. Hence, the following WFR is specified.
WFR: An Actor must have a name [17].

context Actor **inv** :
name − > notEmpty()

4.1.4 *Inherited Association Multiplicity Restriction.*

When the superclass contains an association, its subclasses inherit it with the same multiplicity value. Association multiplicity specified in super-class generally encompass all the possible multiplicity values that the association can take in each sub-class. Necessarily, the inherited multiplicity value then may exceed the accurate value that each sub-class is supposed to have. To specify the correct association multiplicity in sub-classes if it is needed, one needs to specify OCL constraints that reduce the association multiplicity range. In this case, do subclasses inherit the association with the same multiplicity as specified in the superclass?

As illustrated in Fig 7, An ActivityNode has *incomings*. Based on the metamodel structure, all *ActivityNodes* (InitialNode, FinalNode, JoinNode, ForkNode) can have 0 to * incomings. However, UML domain semantics depicts that each Node has its specific number of incomings. For instance, a ForkNode must have one incoming ActivityEdge. The following WFR is then specified.
WFR: A ForkNode has one incoming ActivityEdge [17].

context ForkNode **inv** :
incoming − > size() = 1

4.1.5 *Inherited Attribute Value Restriction.*

In general, in hierarchies, the attribute value range in super-class is not specified, or specified widely through OCL constraints to include all the possible values that the attribute can take in sub-classes. Consequently, to avoid prohibited attribute values with respect to application-domain, OCL constraints restricting the wide values range are specified in sub-classes. The question that we might ask is: are there values in the space of possible values that the attribute should not take?

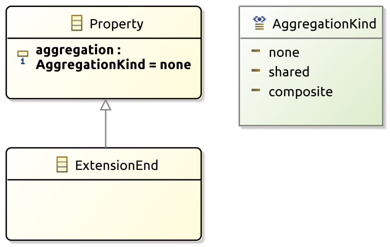

Figure 5: Attribute Value Restriction

Taken the UML structure in Fig 5, the aggregation of an *ExtensionEnd* may take as value one of the three *AggregationKind* literals, namely *none, shared,* or *composite.* With respect to UML semantic, it must take *composite* literal. The following WFR is then specified to avoid semantic incorrect values.
WFR: The aggregation of an ExtensionEnd is composite [17].

context ExtensionEnd **inv** :
self.aggregation = AggregationKind::composite

4.1.6 *Inherited Operation Value Restriction.*

Operations defined in superclass describe common subclasses behaviors. Sometimes, these behaviors may change from a sub-class to another, and need to be more specific in sub-classes. This is done through OCL constraints defined in sub-classes having more specific behavior. We have noticed two types of operations on UML. The first type concerns operations that have classes as their type, and hence returns a set of objects. The second type concerns operations that assess system-state, sometimes based on a set of parameters. The latter operation type is often constrained on a subset of sub-classes instances or to specify the desired result that the operation should return for the given parameters (i.e. the desired system state).

In Figure 6, operation *is(Integer,Integer)* states whether the bounds of the *multiplicityElement* equals the entered operation parameters. Here, multiplicity of the *OutputPin* that results from *CreateLinkObjectAction* must be [1..1]. An OCL constraint specifying this condition is then expressed as depicted below.

WFR: The multiplicity of the OutputPin is 1..1 [17].

context CreateLinkObjectAction **inv** :
result.is(1,1)

4.1.7 *Type Relation.*

Given two classes A and B linked with an association "x", where one or both classes have subclasses, it is possible to create models where not only A instances are linked

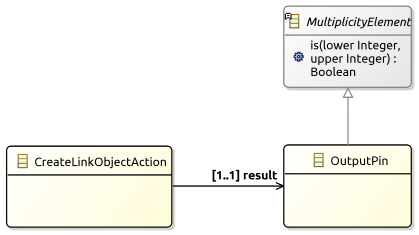

Figure 6: Inherited Operation Value Restriction

with B instances through "x", but also all the possible associations that link A and its subclasses instances with B and its subclasses instances. In some cases, some relationship combinations do not exist in the represented domain, and need to be restricted in the metamodel. Consequently, Restricting nonexistent relations with

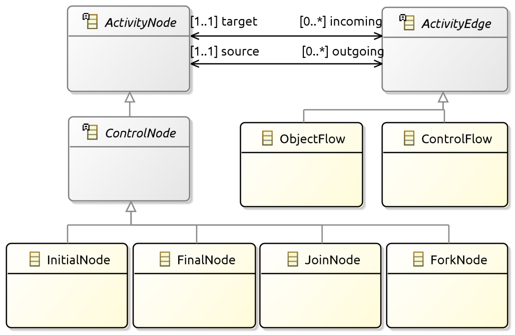

Figure 7: Type Relation

OCL constraints becomes a mandatory step to make metamodel more precise. Based on MM structure in Figure 7 taken from UML Activities MM, *InitialNode* outgoing *ActivityEdges* can be of type *ControlFlow* or *ObjectFlow*, which is incorrect with respect to UML semantics. In fact, an *InitialNode* outgoings can be of type *ControlFlow* only. The following WFR will then complement the structure to specify the type.
WFR: All the outgoing ActivityEdges from an InitialNode must be ControlFlows [17].

context InitialNode **inv** :
outgoing − > forAll (oclIsKindOf (ControlFlow))

4.1.8 *Cycles Restriction.*

In metamodel context, a cycle is a succession of associations and operations wherein a class is reachable from itself. More subtle cycles can also be present, which link a class to its superclass through some navigations (associations and operations that have class as their type). Presence of cycles does not allows auto-association only, but also diamond configurations [35]. Model developers need to be aware of that and need to assess whether these associations are valid with respect to their application domain. Otherwise, OCL constraints that prevent from auto-associations need to be defined.

The cycle constraints are applied to:

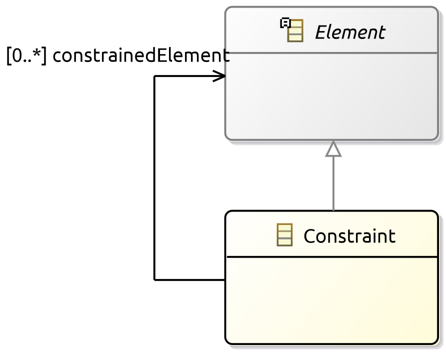

Figure 8: Indirect reflexive association

(1) restrict the reflexive navigation size, or specify whether auto-association must be accepted or not;
(2) compare the attribute value of a class instance with attribute value of its related class instances through the reflexive navigation;
(3) specify an attribute value in the class instances that are related with the "self" class through the reflexive navigation;
(4) restrict an operation result in the related class instances through the reflexive navigation.

Given a cycle, one can ask the following questions:

(1) can a class instance be associated to itself?
(2) do the class instances that are related to "self" through the reflexive navigation have some specificities that must be specified?

For example, according to Figure 8, a constraint is applied to an *Element*. Accordingly, since the *Constraint* is an *Element*, it is possible to apply a constraint to itself. Restricting this indirect cycle becomes then necessary. *WFR*: A *Constraint* cannot be applied to itself [17].

context Constraint **inv** :
not constrainedElement − > includes(self)

4.1.9 *Different Paths Relation.*

Starting from a class A, if we find two distinct navigation successions that lead to the same class B, it is possible that there is a semantic link between them that needs to be made more precise. The model designer needs to make this link explicit by adding OCL constraint to avoid semantic ambiguities. Given this MIS, OCL constraints are written to link the two collections that are obtained from the two navigations with sets operators (inclusion, exclusion, equality, difference), or linking their size. The question related to this MIS is: what is the link between the two distinct paths that link two classes?

As illustrated in Fig. 9, starting from *LinkEndData* class, it is possible to arrive to *InputPin* by passing directly through *value* association, or passing by *QualifierValue* through qualifier association, then value association. Diagrammatically, the link between the two navigations is not explicit. Hence, the following OCL constraint is written to define the link between these two navigations.
WFR: The value InputPin is not also the qualifier value InputPin [17]
.

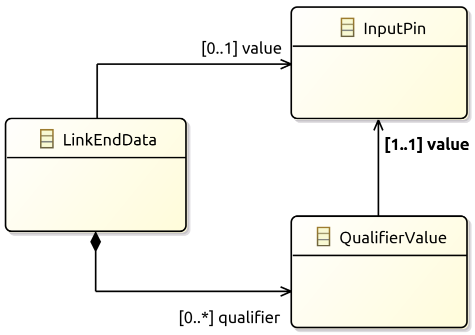

Figure 9: Different Paths Relation

context LinkEndData **inv** :
value − > excludesAll(qualifier.value)

4.2 What is the proportion of the contraints that are applied to complete the identified metamodel inaccurate structures?

After characterizing a set of MIS from the studied metamodel, we need to ensure that these structures typically need to be complemented with constraints to fine-tune their semantics. The aim of this section is to investigate whether the characterized MIS in one of the studied metamodels are also present in the other ones, and most importantly refined with OCL constraints to complete their semantic. To do so, we evaluate the importance of our finding from quantitative perspective by counting the occurrences of each MIS-related constraint.

As shown in Fig. 10, we can see that the MIS-related constraints proportion changes from one metamodel to another. For instance, in Diagram Definition metamodel, we can see that the MIS-related constraints exceeds 90%, while on SAM metamodel, they represent only 17% of the total number of specified OCL contraints. On average, the MIS-related constraints proportion approximates 53%. Table 2 the occurrences of MIS-related constraints. We can see that 414 constraints out of 812 have been added to complete the semantic over these metamodel inaccurate structures, which represents 53% of all the expressed constraints of the studied metamodels. We did notice that some MIS are more refined with OCL constraints than other ones. For instance, the attribute value restriction MIS and the paths are refined in seven out of the ten metamodels. Conversely, the enumeration MIS was found with OCL constraints only in the UML metamodel.

Considering the impossibility to precise in diagrammatic way the values set or range that the attribute may take to respect domain semantic, OCL constraints that are used to restrict the value of attributes are very frequent. Also, *Different Paths Relation MIS* is the most constrained MIS in our list. This is because the absence of constraints that explicit the link between some paths can lead to major inconsistencies that should not appear in a well-formed artifact. For this reason, expliciting all the links between the related concepts must be carried out by metamodel designers through OCL constraints. When it comes to *Cycles*, they have been pointed out previously in many work([5, 35]) mostly for the inconsistencies they lead to. Deciding whether the cycles cause semantic inconsistencies remains mandatory. The occurrence of *Type Relation, Inherited Optional Attribute, Inherited Operation Value Restriction, Inherited Attribute Value Restriction, Inherited Association Multiplicity Restriction* MISs depends on the presence or absence of the inheritance structures. Since almost all the MIS that we previously identified previously are present in at least two metamodels with OCL constraints, we can be sure that their presence in metamodels presents often a lack of precision that must be completed with OCL constraints. Hence, more attention needs to be given by metamodel designers during design or refactoring to complete their semantics if they are not enough accurate. We note that the presence of a MIS in a metamodel does not necessarily mean that this structure does not encompass all semantic it should have. Then, metamodel designer intervention is the only meant to confirm whether it lacks semantic, and hence if constraints need to be expressed to complete semantic. We decided to keep Enumeration literals restriction MIS because we believe that it is not possible to define the set of literals that an attribute of type enumeration can take unless using OCL language.

5 THREATS TO VALIDITY

As for any experimental evaluation, some threats could affect the validity of our findings. For the internal validity, we believe that manually analysing metamodel fragments to find out if they lead to doubt about the existence of OCL constraints is subjective, and depends on the experience of those who analyse. For instance, a designer who has already worked on completing metamodel semantics with OCL constraints will find out more metamodel inaccurate structures than an inexperienced designer or a student. To avoid that, each designer conducted the analysis individually in order to create his/her metamodel inaccurate structures set, then, we proceeded to a vote to decide about each MIS to obtain at the end the presented metamodel inaccurate structures list.

Another threat that could affect the validity of our findings is the data. Indeed, we relied on metamodels that are already refined with OCL constraints. For certain metamodels, the set of OCL constraints might not be complete. We believe that studying many other metamodels is necessary to complete the set of metamodel inaccurate structures. To mitigate this threat, we have analyzed 10 different metamodels.

6 RELATED WORK

The related research is presented in three different perspectives. The first one concerns the assistance in the specification of OCL constraints through OCL patterns. The second perspective is related to the assistance of constraints co-evolution and refactoring. Finally, the last perspective concerns the automatic generation of OCL constraints. The first type of assistance is done through OCL patterns. To the best of our knowledge, the closest work to ours are [33–35] where a method and tools have been provided to develop concise and consistent constraint specifications, and hence assist precise modeling. First, limitations of expressiveness of graphical modeling languages have been captured as "anti-patterns". Then, OCL constraint patterns that correct these anti-patterns have been proposed.

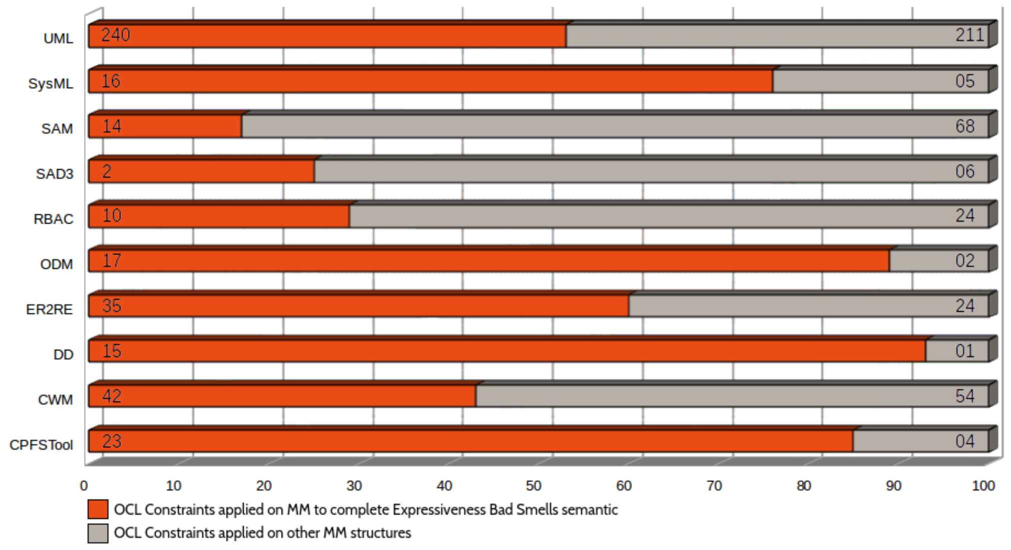

Figure 10: MIS-related constraints proportion for each MM

	UML	SysML	ODM	CWM	DD	SAD3	CPFSTool	ER2RE	RBAC	SAM	**Total**
Type Relation	3	1	14	13	0	0	0	0	0	0	59
Attrib Value Rest	7	0	0	1	15	2	6	13	0	7	51
Enum Lit Rest	2	0	0	0	0	0	0	0	0	0	2
Inh Optional Attr Rest	3	0	3	0	0	0	0	0	0	0	6
Inh Asso Multip Rest	6	0	0	6	0	0	1	0	0	0	67
Inh Attr Value Rest	12	1	0	1	0	0	4	0	0	0	18
Inh Oper Value Rest	23	0	0	0	0	0	0	0	0	1	24
Cycles Restriction	18	6	0	17	0	0	0	0	3	0	44
Different Paths Rest	84	8	0	4	0	0	12	22	7	6	143
Total	240	16	17	42	15	2	23	35	10	14	**414**
MIS-constraints proportion	53%	76%	89%	44%	94%	25%	85%	59%	29%	17%	53%

Table 2: MIS associated with OCL constraints occurrences

The idea behind the anti-patterns in this paper is to point out almost all the situations where OCL constraints can be used. Conversely, we characterize only the structures that lead to a doubt about the expressiveness. Also, a tool that assists designers in expressing constraints has been developed. It implements OCL patterns but does not automatically detect all the anti-patterns occurrences. Then, the metamodel designer is the one who needs to precise where the constraints need to be specified. In our case, MIS indicates the exact fragment potentially needing a constraint, and the designer will have to decide whether the MIS occurrence give place to a lack of expressiveness or not. We think that this work is complementary with ours.

Constraint patterns have been introduced first for object-oriented programming in [23], then adapted to conceptual models and metamodels. For instance, authors of [29, 30] revealed types of constraints relevant for design of well-formed conceptual models in form of taxonomy. This was done through analysis of the most important conceptual modeling methods to identify the situations where these constraints should be used. Then, in [10] they did define a profile that extends the set of UML predefined constraints with some types of constraints that are used very frequently in conceptual schemes. Also, the work in [6, 7] aim at adapting some existing constraint patterns to increase the efficiency of testing and debugging processes. Furthermore, the collection of published constraint patterns has been extended in [18]. Authors of [4] have done an empirical analysis on metamodels with well-formedness rules aiming at understanding how metamodelers articulate both languages, and asserting metamodeling practices in the previous ten years. To conclude, a set of OCL constraint patterns have been identified in [5]. In the contrary of our work, their work was more focused on analysing OCL constraints and their structure, while we used constraints just to capture the targeted MOF structure, and we analysed MOF structures.

The co-evolution of OCL constraints that follows the evolution of metamodels was the center of attention of many research work.

We may distinguish semi-automatic co-evolution methods. For instance, Hassam et al. [20, 21], Khelladi et al. [24, 25] and Kusel et al. [27] propose semi-automatic co-evolution approaches of OCL expressions. On the other hand, other work propose fully automated methods. For example, Cabot et al. [3], Demuth et al. [13, 14], or Markovic et al. [28] proposed an approach in which they formalize the most important refactoring rules for class diagrams and classify them with respect to their impact on annotated OCL constraints. Batot et al. [2] propose an automatic two-steps process to automatically co-evolve metamodels and OCL constraints using genetic algorithm. For refactoring constraints written in OCL, both Correa et al. [8, 9] proposed specifications to improve the understandability of OCL constraints. On the other side, [32] conducted a literature survey, collecting and categorizing several refactoring types, and were implemented as a tool for refactoring. Then Hong et al. [22] proposed an automated search-based OCL constraints refactoring approach. While all the work that co-evolve and refactor OCL constraints rely on metamodel and the existing constraints set to perform updates, we rely only on metamodel structure only. Our approach can be used to add new constraints that did not appear on metamodel old version, but can not be used to evolve existing ones. Many researches have been conducted to Faunes et al. [15] propose an automatic approach to retrieve OCL constraints from a set of valid and invalid model examples and metamodel structure using genetic algorithm. In the same context, Dang et al. [11, 12] infer business rules from user scenarios and OCL patterns using CSP. The major difference between our work and both mentioned above is that we do rely only on metamodel structure, while both use correct and incorrect model examples to extract and exploit all the information that could be extracted to specify them in form of constraints. Automatic inference is a great solution if the set of examples is enough wide and diverse to capture (almost) all the concepts bounds and limits in the space of possible. Otherwise, the generated constraints would not be significant and would be too specific to the used models. We believe that both automatic and manual solutions are efficient, but the choice depends on the context.

7 CONCLUSION

In this paper, a set of metamodel inaccurate structures has been pointed out, which are metamodel structures that are often refined with OCL constraints because they are not enough accurate to precisely capture domain semantic. To do so, an analysis has been done using a set of OCL constraints taken from 10 metamodels to identify these low-accuracy structures, and four metamodel experts have studied each constrained structure in the metamodel. Consequently, a set of metamodel inaccurate structures have been proposed. To ensure that these structures can be found in any other metamodel, further study was carried out to quantify each metamodel inaccurate structure (MIS) occurrence in each of the ten metamodels. We believe that this work could help designers while creating and refactoring metamodels in detecting low-expressive structures that may imply inconsistencies in generated artifacts if not refined with OCL constraints. The main contribution of this work is to point-out the MISs, and also to give quantitative evidences about their impact in metamodels.

We are aware that our dataset does not contain all the possible MISs. So, as a future work we will investigate further metamodels and OCL constraints hopping to discover others MISs. We also envisage to complete this work by assisting metamodel designers with an adequate tool. Indeed, we intend to create a tool that automatically identifies MISs and suggests possible OCL constraints, or refactor these structures to structurally avoid these inaccuracies.

REFERENCES

[1] [n. d.]. The Repository for Model-Driven Development. http://remodd.org/.
[2] E. Batot, W. Kessentini, H. Sahraoui, and M. Famelis. 2017. Heuristic-Based Recommendation for Metamodel — OCL Coevolution. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 210–220. https://doi.org/10.1109/MODELS.2017.25
[3] Jordi Cabot and Jordi Conesa. 2004. Automatic integrity constraint evolution due to model subtract operations. In *International Conference on Conceptual Modeling*. Springer, 350–362.
[4] Juan Cadavid, Benoit Combemale, and Benoit Baudry. 2012. *Ten years of Meta-Object Facility: an analysis of metamodeling practices.* Ph.D. Dissertation. INRIA.
[5] Juan José Cadavid Gómez. 2012. *Assistance à la méta-modélisation précise.* Ph.D. Dissertation. Rennes 1.
[6] Dan Chiorean, Vladiela Petraşcu, and Ileana Ober. 2010. Testing-oriented improvements of OCL specification patterns. In *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, Vol. 2. IEEE, 1–6.
[7] Dan Chiorean, Vladiela Petrascu, and Ileana Ober. 2012. MDE-driven OCL Specification Patterns. *Journal of Control Engineering and Applied Informatics* 14, 1 (2012), 83–92.
[8] A. Correa. 2009. Refactoring to improve the understandability of specifications written in object constraint language. *IET Software* 3 (April 2009), 69–90(21). Issue 2. https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2008.0022
[9] Alexandre Correa and Cláudia Werner. 2007. Refactoring object constraint language specifications. *Software & Systems Modeling* 6, 2 (01 Jun 2007), 113–138. https://doi.org/10.1007/s10270-006-0023-y
[10] Dolors Costal, Cristina Gómez, Anna Queralt, Ruth Raventós, and Ernest Teniente. 2006. Facilitating the Definition of General Constraints in UML. In *Model Driven Engineering Languages and Systems*, Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 260–274.
[11] Duc-Hanh Dang and Jordi Cabot. 2013. Automating inference of ocl business rules from user scenarios. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 1. IEEE, 156–163.
[12] Duc-Hanh Dang and Jordi Cabot. 2015. On Automating Inference of OCL Constraints from Counterexamples and Examples. In *Knowledge and Systems Engineering*. Springer, 219–231.
[13] Andreas Demuth, Roberto E Lopez-Herrejon, and Alexander Egyed. 2012. Automatically generating and adapting model constraints to support co-evolution of design models. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 302–305.
[14] Andreas Demuth, Roberto E Lopez-Herrejon, and Alexander Egyed. 2013. Supporting the co-evolution of metamodels and constraints through incremental constraint management. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 287–303.
[15] Martin Faunes, Juan Cadavid, Benoit Baudry, Houari Sahraoui, and Benoit Combemale. 2013. Automatically searching for metamodel well-formedness rules in examples and counter-examples. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 187–202.
[16] Object Management Group. [n. d.]. https://www.omg.org/.
[17] Object Management Group. [n. d.]. Unified Modeling Language 2.5. https://www.omg.org/spec/UML/2.5/, year = 2015.
[18] Ali Hamie. 2013. Constraint specifications using patterns in OCL. *International Journal on Computer Science and Information Systems* 8, 1 (2013).
[19] Muhammad Hammad, Tao Yue, Shuai Wang, Shaukat Ali, and Jan F Nygård. 2017. IOCL: An interactive tool for specifying, validating and evaluating OCL constraints. *Science of Computer Programming* 149 (2017), 3–8.
[20] Kahina Hassam, Salah Sadou, and Régis Fleurquin. 2010. Adapting ocl constraints after a refactoring of their model using an mde process. In *9th edition of the BElgian-NEtherlands software eVOLution seminar (BENEVOL 2010)*. 16–27.
[21] K. Hassam, S. Sadou, V. L. Gloahec, and R. Fleurquin. 2011. Assistance System for OCL Constraints Adaptation during Metamodel Evolution. In *2011 15th European Conference on Software Maintenance and Reengineering*. 151–160. https://doi.org/10.1109/CSMR.2011.21

[22] Lu Hong, Shuai Wang, Tao Yue, Jan F Nygard, et al. 2017. Automated refactoring of OCL constraints with search. *IEEE Transactions on Software Engineering* (2017).

[23] Bruce Horn. 1992. Constraint patterns as a basis for object oriented programming. In *Proc. ACM OOPSLA*. 218–233.

[24] Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, Jacques Robin, and Marie-Pierre Gervais. 2016. Detecting complex changes and refactorings during (meta) model evolution. *Information Systems* 62 (2016), 220–241.

[25] Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, Jacques Robin, and Marie-Pierre Gervais. 2016. Metamodel and constraints co-evolution: A semi automatic maintenance of ocl constraints. In *International Conference on Software Reuse*. Springer, 333–349.

[26] Anneke Kleppe. 2008. *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education.

[27] Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Werner Retschitzegger, Johannes Schoenboeck, Wieland Schwinger, and Manuel Wimmer. 2015. Systematic co-evolution of OCL expressions. *11th APCCM* 27 (2015), 30.

[28] Slaviša Marković and Thomas Baar. 2005. Refactoring OCL annotated UML class diagrams. In *International Conference On Model Driven Engineering Languages And Systems*. Springer, 280–294.

[29] Elita Miliauskaitė and Lina Nemuraitė. 2005. Representation of integrity constraints in conceptual models. *Information technology and control* 34, 4 (2005).

[30] Elita Miliauskaite and Lina Nemuraite. 2005. Taxonomy of integrity constraints in conceptual models. In *IADIS virtual multi conference on computer science and information systems*. 247–254.

[31] NaoMod. [n. d.]. Metamodel zoos.
https://naomod.github.io/.

[32] Jan Reimann, Claas Wilke, Birgit Demuth, Michael Muck, and Uwe A. 2012. Tool Supported OCL Refactoring Catalogue. In *Proceedings of the 12th Workshop on OCL and Textual Modelling (OCL '12)*. ACM, New York, NY, USA, 7–12. https://doi.org/10.1145/2428516.2428518

[33] Michael Wahler, David Basin, Achim D Brucker, and Jana Koehler. 2010. Efficient analysis of pattern-based constraint specifications. *Software & Systems Modeling* 9, 2 (2010), 225–255.

[34] Michael Wahler, Jana Koehler, and Achim D Brucker. 2007. Model-driven constraint engineering. *Electronic Communications of the EASST* 5 (2007).

[35] Michael S Wahler. 2008. *Using patterns to develop consistent design constraints*. Ph.D. Dissertation. ETH Zurich.