Electronic Communications of the EASST
Volume 36 (2010)

Proceedings of the Workshop on OCL and Textual Modelling (OCL 2010)

Towards a Conceptual Framework Supporting Model Compilability

Dan Chiorean, Vladiela Petraşcu

14 pages

Guest Editors: Jordi Cabot, Tony Clark, Manuel Clavel, Martin Gogolla
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards a Conceptual Framework Supporting Model Compilability

Dan Chiorean[1], **Vladiela Petraşcu**[2]

[1]chiorean@cs.ubbcluj.ro
[2]vladi@cs.ubbcluj.ro
Faculty of Mathematics and Computer Science, Computer Science Department
Babeş-Bolyai University, Cluj-Napoca, Romania

Abstract: The ever-growing use of modeling languages today is largely due to a maturation of model-based development technologies. However, there is enough room for improving language specifications and consequently, the efficiency of their usage. The state of facts in specifying Well Formedness Rules is among the most important issues calling for improvements. Despite the fact that various papers have approached it, the topic is still open. To solve it, there is the need of a rigorous conceptual framework supporting the specification of modeling languages' static semantics. This would stand as a basis for ensuring model compilability, a mandatory requirement in a model-driven context. Through this paper, we aim at providing core ideas that would contribute to the creation of such a framework. Our approach is testing-oriented and promotes the use of OCL specification patterns.

Keywords: model compilability, metamodel, WFRs, static semantics, MOF, UML

1 Introduction

The value of both correctness and completeness in a language specification is widely acknowledged. A poor language definition negatively impacts every model employing the language in question. The emergence of the model-driven paradigm[1] is grounded on significant changes in the model usage requirements, triggering counterparts in the area of modeling languages and technologies. These concern an increased rigor of language definitions, accompanied by an alignment of technologies to the specifics of the model-driven development process.

The automation goals envisioned by the aforementioned paradigm call for appropriate tools. In a model-driven context, models are equally used as inputs and outputs of such tools. This places two critical requirements on them: (1) being defined in rigorous languages supporting their efficient manipulation, and (2) being fully compliant to (correct with respect to) those languages. Well Formedess Rules (WFRs) play a key role in ensuring the completeness and rigor of a language definition and stand as a basis for assessing the correctness of models with respect to it.

Researches in the field have revealed that modeling and programming languages share more commonalities than differences [CSW08]. The differences are mainly related to a higher abstraction level employed by modeling languages, as well as to the common use of graphical

[1] This paradigm comes in various flavors, such as MDA (Model Driven Architecture), MDE (Model Driven Engineering), MDD (Model Driven Development) or LDD (Language Driven Development).

formalisms for representing their concrete syntax. The graphical formalism though is exclusively meant to support understandability, as all model processing tasks (serialization included) employ a textual syntax representation. Moreover, nowadays there are experts promoting the use of a textual concrete syntax even for model specification[2] [SS08], [FK]. Acknowledging the truth that the graphical and textual concrete syntaxes are equivalent, that modeling and programming languages are closely similar, assists in accepting the fact that, similar to program compilability, full compliance of a model with its modeling language is a must.

Models that conform to their modeling language's WFRs are generally referred to as *well formed models* in the literature. However, we propose using the phrase *compilable models* instead of *well formed models*. The arguments are twofold. On the one side, aparts from WFRs, there may be other kinds of rules that a model has to comply with, such as methodological rules, metric rules, or business rules. A *well formed model* should designate a model complying with any of these rules, which is a generic requirement compared to mere compilability. On the other side, the newly proposed phrase stresses on the similarity among modeling and programming languages, hence on the imperative nature of the compilability requirement.

Therefore, regardless of their size, models' compilability is a must if we want them manipulated by tools. While in case of small (didactical) ones, checks may even be manually performed, in case of medium-sized to large models the existence of appropriate verification instruments becomes a mandatory requirement.

The fact that models, at their turn, will generally be complemented by assertions denoting different kinds of business rules is just another argument for requiring enforcement of model compilability. An uncompilable model may trigger failures in the compilation of its own assertions [3].

Despite all these arguments, current practice shows that model compilability is more a goal than a reality. This state of facts has both human and technological roots. On the one side, there is the unfortunate assumption that seems to be still governing the developers' community (worse, even the researchers' one), according to which models are primarily meant to facilitate problem understanding and assist the client-developer communication, a rigorous model verification not being therefore an imperative. On the other, there are the shortcomings concerning the formalisms and tools involved in compilability assessments. The current paper aims at identifying the reasons underneath this state of facts, with the intention of proposing solutions meant to improve it.

The rest of the paper is organized as follows. Section 2 diagnoses the state of facts in the field of model compilability, and summarizes our contribution as compared to related approaches. The principles that we propose at the basis of a conceptual framework meant to support an accurate specification of modeling languages' static semantics (a prerequisite in enforcing model compilability) are explained in Section 3. A proof of concepts is provided in Section 4, by means of two relevant examples of UML WFRs. The paper ends with conclusions and hints on future work in Section 5.

[2] However, even in this case, the language used for model representation is complemented by a second language, targeted at model navigation and specification of constraints.

[3] As an example, the existence of several equally named properties in a class will make an OCL expression referring to any of these properties uncompilable.

2 Model Compilability - Reality and Goals

2.1 Diagnosing the State of Facts

As previously pointed out, there is a strong technological factor involved in what causes the current state of facts in the field of model compilability. As a proof, we cite two recent references arguing the behavior of a few UML2 tools and of ArgoUML when required to perform model validation.

In a posting entitled "Poor validation of UML Models in Eclipse UML2 tools" [por], the author analyzes the failure reasons of an XMI transformation service, concluding that the failure has been caused by the use of an Eclipse UML2 not well-formed model. In [BF09], the authors report on identifying "counter examples" of UML models (models breaking well-formedness rules) that were not catched by the ArgoUML tool. The former reference highlights the fact that the Eclipse UML2 tools under study enable the creation of uncompilable UML2 models, while the latter raises a warning with respect to the proper translation of WFRs into Java. In addition, [por] reveals the fact that the tools under study fail to implement the entire set of WFRs.

However, the problems with the aforementioned UML tools are only the visible tip of the iceberg. In fact, these problems are rooted in the absence of a general consensus within the modeling community with respect to the necessity and means of using constraints. Consequently, the real, "hidden" issues are related to the inadequate specification, deficient validation, and poor understanding of constraints and their usage. This statement is motivated by a detailed analysis concerning the specification and use of constraints within UML and some of the best known meta-metamodels (MOF [OMG06], Ecore [SBPM08], and XCore [CSW08]). Unfortunately, a number of specification errors reported for the UML 1.x WFRs ([RG00], [CCBC04]) have not been fixed yet, being further inherited by the MOF 2.0 and UML 2.x documents. These problems have significantly affected the possibility of checking models' compilability, and have even compromised the concept of *model compilability* itself.

2.2 Related Work

Through the last decade, a significant number of papers focusing on the specification and usage of WFRs have been written. We claim that [RG00], [FQL+03] and [CCBC04] are the closest to the approach presented in this paper. Thus, we briefly summarize their contents in the following.

In [RG00], the authors have given a first quasi-exhaustive analysis of the WFRs specified in UML 1.3. The work has focused on the Foundation::Core package (31 classes and 27 associations) that has been specified in USE, in order to check the corresponding 43 WFRs. Also, 28 Additional Operations were tested. Errors have been found in 39 out of 71 tested expressions. Four categories of errors have been identified: syntax errors, minor inconsistencies, type checking errors, and general problems. The paper was the first to draw an alarm with respect to the quality of the UML WFRs specifications. The following statement worth mentioned: "For future work we plan to extend the analysis to the complete UML metamodel including all of its well-formedness rules and making it available in USE. This might not only be usefull for improving the state of the standard but also implies another very nice application: in principle, any UML model can be checked for conformance to the UML standard. "

In [FQL+03], the second published paper on this topic, the authors claim having tested the entire set of WFRs specified in the context of the UML 1.3 metamodel. They report 450 errors of three kinds: non-accessible elements, empty names, and miscellanea. The proposed solutions for fixing the reported problems seem a bit bizarre. Namely, they suggest to "Take the empty names into account in every rule of the metamodel (296 errors). Consider access and contents as two different concepts (138 errors). Avoid two opposite association ends with the same name (18 errors)".

In [CCBC04], the authors analyze different kinds of errors and propose new specifications to fix the identified bugs. As the title suggests, the focus is on proposing "good practices" meant to support "a correct, clear and efficient specification". The consistency among the formal and informal specifications, the clearness of OCL expressions, the fact that evaluating OCL specifications instead of only compiling them is imperative, are among the proposed and exemplified practices.

2.3 Setting the Goals

As shown in the previous subsection, there are various papers signaling the inadequacy of UML WFRs specifications. However, most of the reported work has focused on the uncompilability of WFRs with respect to OCL. Still, a closer look at the standard specifications (both WFRs and Additional Operations (AOs)) reveals that, apart from compilability issues, the specifications in question enclose logical errors, as well as drawbacks caused by their superficial testing[4].

In this context, the current paper aims at contributing to the set up of a framework supporting an accurate definition of the static semantics of modeling languages and enabling efficient model compilability checks. The topics addressed outrun those concerning the mere compilability of OCL WFRs, being closer to the conceptual issues involved in specifying a static semantics. Namely, we focus on the difference between WFRs and "classical invariants" and the importance of choosing an appropriate specification context, test-driven specifications, testing-oriented specifications and the use of OCL specification patterns.

The experience aquired while working at and with the UML 1.5 WFRs in OCLE [LCI] has allowed us to conclude that the task of creating an efficient OCL specification for the static semantics of an UML metamodel is not an easy one, even for specialists. This may explain the previously reported state of facts in the field. Therefore, explaining the basics of the specification approach that we promote and arguing it by means of selective examples, is believed to provide greater benefits than merely depicting the complete, final set of WFRs. Moreover, the chosen examples (one related to the semantics of composition, and the other to the name uniqueness constraint within namespaces) have not been randomly picked; they are both well known, core metamodeling issues. The fact that even their corresponding specifications are bogus is a strong argument towards the adoption of a rigorous WFRs specification framework, as promoted by this paper.

[4] The information outputed in case of an invariant violation is insufficient for error diagnosis.

3 Requirements of a Static Semantics Specification

Compilability of a model is checked against its metamodel and associated WFRs; the metamodel defines the abstract syntax of the modeling language, while the WFRs enclose its static semantics. In order to fully serve its intended purpose of supporting efficient model compilability checks, there are a number of requirements that any set of WFRs should comply with.

The first one is *completeness*; the WFRs should entirely cover the static semantics rules of the language. This entails an intimate understanding of all metamodel-level concepts and how they may be suitably related.

Furthermore, each WFR specification should fulfill some quality criteria. The following three are among the most important, the first two being also among the least addressed in the literature.

1. *Detailed, test-driven informal specification.* Preceding the formal WFR expression with a detailed and rigorous informal equivalent is the basic requirement for ensuring correct understandability of the rule. At its turn, the informal specification should be based on meaningful test snapshots needed for its validation (both positive and negative). By analogy to the programming approach known as *test-driven development*, this *test-driven specification* approach provides for a deeper reasoning with respect to the rules, with a positive effect on the correctness/comprehensiveness of their final statements. In fact, all good programming habits remain valid in the design of sizable OCL specifications.

2. *Testing-oriented formal specification.* The OCL WFRs should be stated so as to facilitate efficient error diagnosis in case of assertion failure. In this respect, [CPO10] argues on the use of several testing-oriented OCL specification patterns. This quality requirement comes from acknowledging the ultimate purpose of models and assertions within a model-driven development process.

3. *Correct and efficient formal specification.* The correctness of an OCL WFR encompasses two different aspects: correctness with respect to its informal equivalent, as well as correctness with respect to the language specification. The former asks for a full conformance between the OCL specifications and their natural language counterparts; the latter enforces compilability, therefore conformance to the OCL standard.

Another aspect to consider when specifying WFRs refers to *choosing the most appropriate context* and shape for each. This involves understanding the differences between a WFR and a "classical invariant", as introduced by object oriented programming (OOP) techniques. Specifically, in OOP, the semantics of invariants states that the invariant of a class should refer exclusively to relationships between the values of its attributes [Mey97],[MO94]. In case the type of the attribute is a reference or a collection of references, the invariant is only allowed to constrain their existence and cardinality, being denied any access to the state of the objects attached to the references. This comes from the fact that objects should be autonomous and have exclusive control over their state. When specifying WFRs however, this rule is seldom obeyed. Generally, the invariant corresponding to a WFR refers to the state of the objects that are accessible by navigation starting from the contextual instance `self`. This semantic difference among WFRs and the "classical" OOP invariants influences the choice of their specification context, their complexity and evaluation.

4 Proof of Concepts

This section is aimed at offering a proof of concepts by means of two case studies centered around two relevant constraints for UML: one related to composition, and the other to name clashes within namespaces.

The first case study reveals the incompleteness of the WFRs set enclosing the semantics of composition in both UML 1.x and 2.x, and emphasizes some inconsistencies among the informal statements and the OCL WFRs related to composition in UML 2.x. The proposed solutions stem from an analysis supported by the use of the *test-driven specification* principle. The possibility of expressing the same informal constraint in different contexts and under different shapes, as well as the criteria involved in choosing the right ones are also discussed and exemplified here.

The second case study uncovers three types of errors within the WFR and AOs prohibiting name clashes within namespaces: syntactic errors, logical ones, as well as faults coming from failure to provide the information required for error diagnosis in case the assertion gets violated. The solution proposed for the latter case involves the use of an appropriate OCL specification pattern.

To sum up, the considered case studies cover all aspects discussed in the previous section. Moreover, apart from proving our point, the solutions offered contribute to improving the static semantics of the UML 1.x and 2.x metamodels.

4.1 On the UML Composition Relationship

Let us consider the UML composition relationship. As inferable from the OMG documents ([OMG05], [OMG10]) and papers such as [BHB+01], composition is a stronger form of association, whose semantics may be captured by the following constraints:

[C1]. Only binary associations can be compositions.

[C2]. At most one end of an association may specify composition (a container cannot be itself contained by a part).

[C3]. An association end specifying composition must have an upper multiplicity bound less or equal to one (a part is included in at most one composite at a time).

[C4]. Since the composite has sole responsibility for the disposition of its parts, the parts should be accessible starting from the container (navigation from container to parts should be enforced).

The above mentioned rules are equally important in defining the semantics of composition and should be all formalized at the metamodel level by means of appropriate WFRs.

In accordance with the *test-driven specification* principle, let us consider the example models from Figure 1. The first (from top to bottom) is correct with respect to the semantics of compostion, as expressed by constraints [C1] to [C4]. The last two are both wrong; the second breaks the [C3] constraint (having an upper bound of 2 on the composition end), while the third violates the navigability constraint [C4] (allowing exclusively a part-to-container navigability).

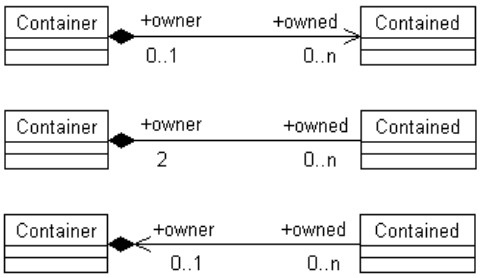

Figure 1: Sample models involving composition

As shown in Figure 2, in UML 1.x, an `Association` is defined by its two `AssociationEnds`. Composition can be specified by setting the `aggregation` enumeration attribute of `AssociationEnd` to `#composite`[5].

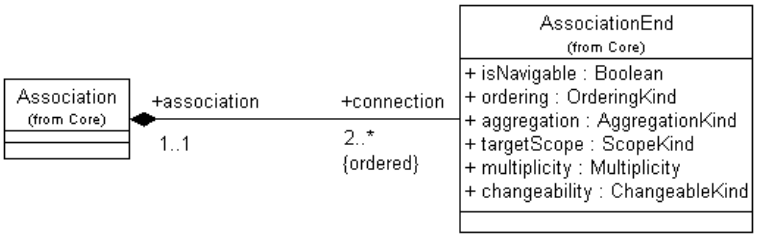

Figure 2: UML 1.4 metamodel excerpt illustrating `Associations`

With respect to enforcing the composition semantic rules [C1] to [C4], the specification only covers the first three of them. The OCL WFRs for [C1] and [C2] are stated in the context of `Association`, while the one for [C3] is written in the context of `AssociationEnd`, as follows:

```
self.aggregation = #composite implies self.multiplicity.max = 1
```

Listing 1: The UML 1.4 WFR for C3

The navigability constraint [C4] is missing from the UML 1.x specification, therefore the third sample model of Figure 1, although incorrect, would be reported as compilable. There are at least three different ways of writing this missing WFR in OCL, as shown below. Favoring one over another is a decision that depends on both language semantics and available tool facilities.

```
context AssociationEnd
 inv validCompositionNavigability1:
  self.aggregation = #composite implies self.association.connection->any(ae |
                                        ae <> self).isNavigable

context AssociationEnd
 inv validCompositionNavigability2:
```

[5] For denoting Enumeration Type values, we kept the notation used in the UML 1.4 specification (`#enumerationLiteral`), instead of `Classifier::enumerationLiteral`, as used in UML 2.x.

```
  self.association.connection->exists(ae | ae <> self and
   ae.aggregation = #composite) implies self.isNavigable

context Association
 inv validCompositionNavigability3:
  self.connection->exists(ae | ae.aggregation = #composite) implies
  self.connection->any(ae | ae.aggregation <> #composite).isNavigable
```

Listing 2: Proposed WFR expressions for C4 in MOF and UML 1.x

The first two invariants from Listing 2 are both written in the context of `AssociationEnd`. If we were to judge from a classic invariants perspective, the second is better, since, in case of assertion failure, the objects which own the slot whose value has caused the failure (`isNavigable`) would be the ones reported as guilty. The first WFR reports the opposite ends. Nevertheless, with the aid of an OCL-supporting tool that allows the evaluation of subexpressions (such as [LCI]), the other ends can be easily accessed.

The third invariant is written in the context of the `Association` metaclass. This specification is the only one fully complying with the UML 1.x composition semantics, stating that the ends of a composition association are both created and destroyed simultaneously with their owning association. According to this, the WFR in Listing 1 can be itself rephrased in the `Association` context as follows.

```
context Association
 inv validCompositionUpperBound:
  self.connection->exists(ae | ae.aggregation = #composite) implies
  self.connection->any(ae | ae.aggregation = #composite).multiplicity.max = 1
```

Listing 3: Proposed WFR for C3 in MOF and UML 1.x

The WFR from Listing 3 and the last WFR from Listing 2 may also be combined within a single OCL expression, as shown below. However, this has the disadvantage of requiring partial evaluation in case of assertion failure, so as to identify precisely which expression in the conjunction has caused the failure.

```
context Association
 inv validCompositionUpperAndNavigability:
  self.connection->exists(ae | ae.aggregation = #composite) implies
  ( self.connection->any(ae | ae.aggregation = #composite).multiplicity.max = 1
    and self.connection->any(ae | ae.aggregation <> #composite).isNavigable )
```

Listing 4: Proposed WFR covering both C3 and C4 in MOF and UML 1.x

As illustrated by Figure 3, the UML 2.x Infrastructure brings some changes in the definition of associations, changes that are also reflected in the MOF 2.0 specification. At the core of these changes stands the removal of the `AssociationEnd` metaclass, and its replacement with `Propety`, "... associated with an `Association` via `memberEnd` attribute" [OMG06] (pp. 66). Regarding navigability, [OMG10] (pp. 112) states that: "An end property of an association that is owned by an end class or that is a navigable owned end of the association indicates that the association is navigable from the opposite ends, otherwise the association is not navigable from the opposite ends."

Unfortunately, concerning the semantics of composition, things seem to have worsened compared to the 1.x specifications. From those four constraints expressing the semantics of composition stated at the beginning of this section, only [C1] has a correct OCL equivalent within

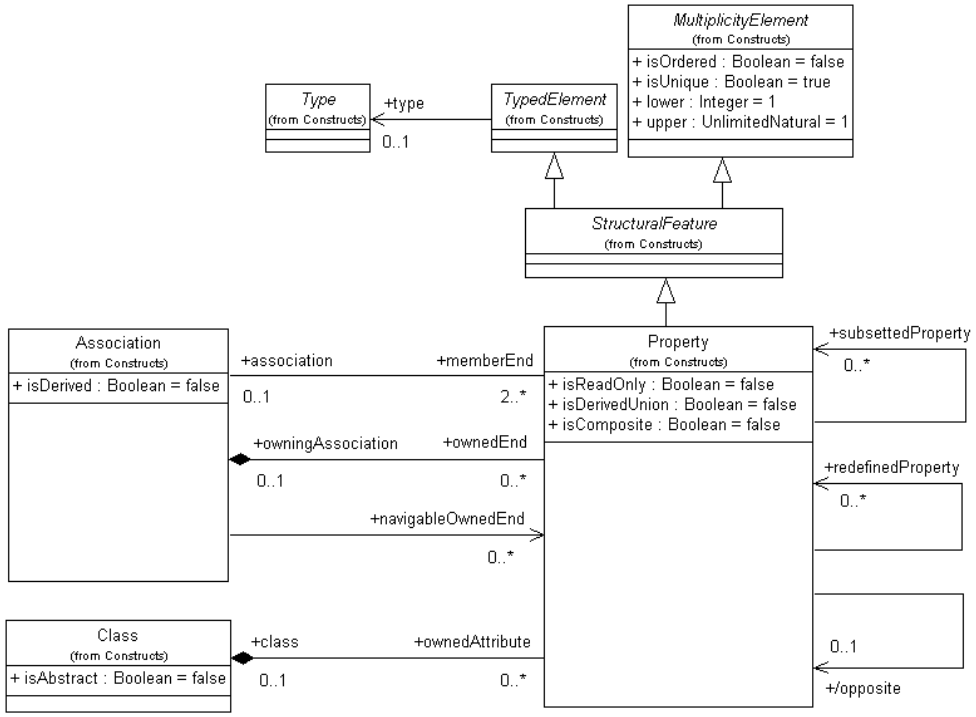

Figure 3: MOF 2.0 and UML 2.3 metamodel excerpt

the specification documents. As for the others, [C4] seems to be missing, [C3] has a drawback that we will detail in the following, and [C2] appears in the MOF 2.0 specification rather as an informal precondition of the `create` operation from the `Reflection::Factory` package.

Regarding composition, [OMG10] (pp. 113) states that "Composition is represented by the `isComposite` attribute on the part end of the association being set to `true`". Given the fact that the word *composite* has a similar meaning to *container*, the previous statement is totally counter-intuitive. It basically reads as *A part in a composition is a composite/container*. In this respect, the OMG specifications should adopt a solution inspired by the EMF Ecore implementation, which has introduced the attributes `container` and `containment` with their natural interpretation.

Overpassing the language ambiguity problem, the OCL WFR corresponding to constraint [C3] found in [OMG10] (pp.125) in the context of Property

```
isComposite implies (upperBound()->isEmpty() or upperBound() <= 1)
```

contradicts the above cited specification statement. If `isComposite` is `true`, then (in accordance with the above) the property plays the role of a part in a composition. Thus, this OCL expression constrains the upper bound of the part, instead of constraining the upper bound of its container.

Given the conflicting situation, we assume the textual statement at pp. 113 of [OMG10], although counter-intuitive, as being the intended one. In tis context, in the following we propose

appropriate OCL WFRs for each of the constraints [C2] to [C4].

The natural context for [C2] is represented by the `Association` metaclass. Its corresponding OCL invariant is given below.

```
context Association
 inv atMostOneCompositeEnd:
  self.memberEnd->select(p | p.isComposite)->size() <= 1
```

Listing 5: Proposed WFR for C2 in MOF and UML 2.x

The rules [C3] and [C4] can be stated both in context of `Association` and `Property`, as shown in Listing 6 and Listing 7.

```
context Association
 inv validCompositionMultiplicity1:
  self.memberEnd->exists(p | p.isComposite) implies
  self.memberEnd->any(p | not p.isComposite).upper = 1

context Property
 inv validCompositionMultiplicity2:
  self.isComposite and self.association->notEmpty() implies
  self.association->any(p | p <> self).upper = 1
```

Listing 6: Proposed WFRs for C3 in MOF and UML 2.x

```
context Association
 inv validCompositionNavigability1:
  self.memberEnd->exists(p | p.isComposite) implies
  self.memberEnd->any(p | p.isComposite).isNavigable()

context Property
def: isNavigable() : Boolean =
(self.class->notEmpty()) xor
(self.owningAssociation->notEmpty() and
 self.owningAssociation.navigableOwnedEnd->includes(self))

context Property
 inv validCompositionNavigability2:
  self.isComposite and self.owningAssociation->notEmpty() implies
  self.owningAssociation.navigableOwnedEnd->includes(self)
```

Listing 7: Proposed WFRs for C4 in MOF and UML 2.x

4.2 On Forbiding Name Clashes within Namespaces

The rule prohibiting name conflicts within namespaces is among the most important WFRs, therefore, in the following we will argue on its specification. In [OMG10], a namespace is defined as follows: "A namespace is an element in a model that contains a set of named elements that can be identified by name." It logically follows that the coexistence under the same namespace of at least two elements having identical names should be forbidden. The type of the elements is irrelevant. In fact, this is the only constraint specified in the `Namespace` context of `Core::Abstractions` (see [OMG10], pp. 73). Its corresponding informal specification states that: "All the members of a Namespace are distinguishable within it." Below is the corresponding formal specification.

```
context Namespace
 inv distinguishableName: membersAreDistinguishable()
```

The `membersAreDistinguishable()` operation is formaly defined as:

```
context Namespace
 def: membersAreDistinguishable():Boolean =
  self.member->forAll( memb | self.member->excluding(memb)
      ->forAll(other | memb.isDistinguishableFrom(other, self)))
```

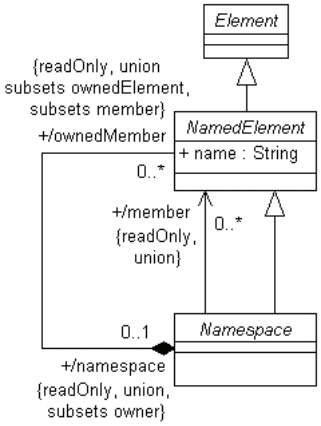

Figure 4: The elements defined in the `Namespace` package

Let us assume that the namespace contains a large number of elements. If this additional operation evaluates to `false`, it is important to discover the identity of those elements producing the failure. With this aim, we propose instead the specification below, which is an instantiation of the `ForAll_Reject` OCL specification pattern proposed in [CPO10].

```
context Namespace
 def: membersAreDistinguishable():Boolean =
  self.member->reject(memb | self.member->excluding(memb)->
   reject(other | memb.isDistinguishableFrom(other, self))->isEmpty())->isEmpty()
```

Both the standard specification and the proposal above employ the AO `isDistinguishableFrom(p1,p2)`. This operation is firstly defined within the `NamedElement` context, being redefined in the `BehavioralFeature` context. As stated in the [OMG10] (pp. 72), the query "... determines whether two NamedElements may logically co-exist within a Namespace. By default, two named elements are distinguishable if (a) they have unrelated types or (b) they have related types but different names."

```
context NamedElement::isDistinguishableFrom(n:NamedElement,ns: Namespace):Boolean
 def: isDistinguishableFrom(n:NamedElement,ns:Namespace):Boolean =
  if self.oclIsKindOf(n.oclType) or n.oclIsKindOf(self.oclType)
   then ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->isEmpty()
   else true
  endif
```

A simple analysis of this last additional operation specification suggests that this query is wrong. In both EMOF and CMOF, `Class` and `Enumeration` are unrelated types. Let `Colour` be the name of two instances, one of the `Class` metaclass (`self`), and the other of the `Enumeration` metaclass (`n`). In this case, the `if` condition evaluates to `false`, and the entire `if` statement to `true`. This evaluation result, allowing the two instances to co-exist in the same namespace, is obviously a wrong one.

As concerning the query `getNamesOfMember(m:NamedElement)`, [OMG10] states that "The query `getNamesOfMember()` gives a set of all of the names that a member would have in a `Namespace`. In general, a member can have multiple names in a `Namespace` if it is imported more than once with different aliases. Those semantics are specified by overriding the `getNamesOfMember` operation. The specification here simply returns a set containing a single name, or the empty set if no name."

```
context Namespace
 def: getNamesOfMember(element:NamedElement): Set(String)=
  if member->includes(element)
   then Set{}->including(element.name)
   else Set{}
  endif
```

The above specification is not compilable, because the type of `Set{}` is `Set(OclUndefined)` and not `Set(String)`. In order to fix the bug, `Set{}` must be replaced with `oclEmpty(Set(String))`.

Behavioral features are namespaces. In this case, the coexistence relationship requests that each two `BehavioralFeatures` have different signatures. Therefore, the query `isDistinguishableFrom()` from specified in the `NamedElement` context must be overriden. In [OMG10], the corresponding formal specification is:

```
context BehavioralFeature
 def: isDistinguishableFrom(n:NamedElement,ns: Namespace):Boolean =
  if n.oclIsKindOf(BehavioralFeature)
  then if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))
                    ->notEmpty()
       then Set{}->include(self)->include(n)->isUnique(bf |
                                        bf.ownedParameter->collect(type))
       else true
       endif
  else true
  endif
```

This specification is not compilable because the type of `Set{}->include(self)->include(n)` is `Set(NamedElement)`, thus `bf.ownedParameter` cannot be computed since `bf` is a `NamedElement`. Following, is the correct specification.

```
context BehavioralFeature
 def: isDistinguishableFrom(n:BehavioralFeature,ns:Namespace):Boolean =
  if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->notEmpty()
  then oclEmpty(Set(BehavioralFeature))->including(self)->including(n)
             ->isUnique(bf | bf.ownedParameter->collect(type))
  else true
  endif
```

5 Conclusions and Future Work

Switching the developers way of thinking with respect to the role of models in software development from *supporting communication and problem understanding* to *supporting code generation and system testing* is a key aspect in ensuring the success of model-driven methodologies. In this respect, we have proposed thinking about model compilability by similarity to program compilability. In this context, the completeness and correctness of WFRs specifications represents a mandatory requirement. Unfortunately, the state of facts is far from expectations. There are no MOF or UML 2.x editors supporting model compilation. In order to surpass this situation, the first priority is fixing the bugs in the existing specifications, followed by the addition of new AOs and constraints, so as to complete the specification of the static semantics.

The approach proposed in this paper highlights the importance of a detailed analysis, reflected in both informal and formal specifications. The use of the *test-driven* specification principle (describing test models prior to writing the formal specification itself) has proven to be a useful technique. Choosing the right context for each WFR is another key issue. This is due to the fact that, as opposed to "classical" invariants, the WFRs may refer to the state of those objects connected to the current object through links. Another core issue concerns the specification style. We have proposed a specification style supporting model testing, by providing the information needed for error diagnosis in case of assertion failure. This testing-oriented specification style relies on the use of appropriate OCL specification patterns [CPO10].

In future, we intend to extend our work to encompass the whole MOF 2.0 and UML 2.3 specifications, as well as to identify and test new specification patterns.

Acknowledgements: This work was supported by CNCSIS-UEFISCSU, project number PNII-IDEI 2049/2008. Our kind thanks go to the anonymous reviewers who's comments helped us in restructuring the paper.

Bibliography

[BF09] P. Bunyakiati, A. Finkelstein. The Compliance Testing of Software Tools with Respect to the UML Standards Specification - the ArgoUML Case Study. In *Proceedings of the Fourth International Workshop on Automation of Software Test (AST'09)*. Pp. 138–143. IEEE CS Press, 2009.

[BHB+01] J.-M. Bruel, B. Henderson-Sellers, F. Barbier, A. Le Parc, R. France. Improving the UML Metamodel to Rigorously Specify Aggregation and Composition. 2001. Technical Report.

[CCBC04] D. Chiorean, D. Corutiu, M. Bortes, I. Chiorean. Good Practices for Creating Correct, Clear and Efficient OCL Specifications. In Koskimies et al. (eds.), *Proceedings of the 2nd Nordic Workshop on the Unified Modeling Language (NWUML2004)*. TUCS General Publications 35, pp. 127–142. Turku Center for Computer Science (TUCS), Finland, 2004.

[CPO10] D. Chiorean, V. Petraşcu, I. Ober. Testing-Oriented Improvements of OCL Specification Patterns. In *Proceedings of the 2010 IEEE International Conference on Automation, Quality and Testing, Robotics - AQTR*. Volume II, pp. 143–148. IEEE Computer Society, 2010.

[CSW08] T. Clark, P. Sammut, J. Willans. *Applied Metamodeling: A Foundation for Language Driven Development*. Ceteva, second edition, 2008. http://itcentre.tvu.ac.uk/~clark/docs/Applied%20Metamodelling%20%28Second%20Edition%29.pdf.

[FK] P. Friese, B. Kolb. Validating Ecore models using oAW Workflow and OCL. In *Eclipse Summit Europe 2007*. http://www.eclipsecon.org/summiteurope2007/presentations/ESE2007_Model-Friese-oAWAndOCL.pdf.

[FQL+03] J. M. Fuentes, V. Quintana, J. Llorens, G. Génova, R. Prieto-Díaz. Errors in the UML metamodel? *ACM SIGSOFT Software Engineering Notes* 28(6):3–3, 2003.

[LCI] LCI (Laboratorul de Cercetare în Informatică). Object Constraint Language Environment (OCLE). http://lci.cs.ubbcluj.ro/ocle/.

[Mey97] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, second edition, 1997.

[MO94] J. Martin, J. Odell. *Object-Oriented Methods: A Foundation*. Prentice-Hall, 1994.

[OMG05] OMG (Object Management Group). Unified Modeling Language (UML) Specification, Version 1.4.2. 2005. http://www.omg.org/spec/UML/ISO/19501/PDF/.

[OMG06] OMG (Object Management Group). Meta Object Facility (MOF) Core Specification, Version 2.0. 2006. http://www.omg.org/spec/MOF/2.0/PDF.

[OMG10] OMG (Object Management Group). Unified Modeling Language (UML), Infrastructure, Version 2.3. 2010. http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/.

[por] MOdeling LAnguages portal. http://modeling-languages.com/blog/content/poor-validation-uml-models-eclipse-uml2-tools.

[RG00] M. Richters, M. Gogolla. Validating UML models and OCL constraints. In Evans et al. (eds.), *UML 2000 The Unified Modeling Language. Advancing the Standard: Third International Conference Proceedings*. Lecture Notes in Computer Science 1939, pp. 265–277. Springer, 2000.

[SBPM08] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, second edition, 2008.

[SS08] M. Seifert, R. Samlaus. Static Source Code Analysis using OCL. In Cabot et al. (eds.), *Proceedings of the 8th International Workshop on OCL Concepts and Tools (OCL 2008) at MoDELS 2008*. Electronic Communications of the EASST 15, p. 15 pages. European Association of Software Science and Technology (EASST), 2008. http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/174/171.