STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 3, 2010

TOWARDS IMPROVING THE STATIC SEMANTICS OF XCORE

VLADIELA PETRAŞCU AND DAN CHIOREAN

Abstract. Throughout this paper, we analyse the state of facts concerning the static semantics of the XCore meta-metamodel and we propose improvements in formalizing it.

1. Introduction

Nowadays, the model-driven techniques in their various flavors (Model Driven Architecture (MDA) [8], Model Driven Engineering (MDE) [11], Language Driven Development (LDD) [5]) promise to revolutionize software development, by automating a major part of the process. Metamodeling languages stand at the core of this novel paradigm. Therefore, a complete and formal definition of these languages (including their abstract syntax, concrete syntax, and semantics) is vital to the success of the model-driven approach.

However, a study that we have carried out on three of the best known meta-metamodels, namely MOF (Meta Object Facility) [7], Ecore [12], and XCore [5], has revealed that the problem of formalizing the static semantics of these languages is far from being a solved issue. Within this paper, we focus on the state of facts regarding the XCore meta-metamodel and we propose improvements in defining its static semantics.

The rest of the paper is organized as follows. Section 2 provides some background on (meta)modeling and identifies the key requirements in defining the static semantics of a (meta)modeling language. A brief overview of

Received by the editors: August 5, 2010.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification – *Programming by contract, Class invariants, Validation*; D.2.11 [**SOFTWARE ENGINEERING**]: Software Architectures – *Languages (e.g., description, interconnection, definition)* .

Key words and phrases. metamodeling, XCore, static semantics, Well Formedness Rules (WFRs).

XCore, the current approach in describing its static semantics, as well as details regarding our contribution are given in Section 3. The paper ends with conclusions in Section 4.

2. Backgound on (meta)modeling

Similar to any other language, a modeling language can be defined as a 3-tuple of the form (abstract syntax, concrete syntax, semantics) [5]. The abstract syntax defines the vocabulary of concepts employed by the language, together with their structural inter-relationships. It has been traditionally given in terms of a class model - called the metamodel[1], which can be visualized by means of class diagrams. Defining a metamodel is an analogous process to defining the BNF (Backus-Naur Form) grammar of a programming language. Moreover, similar to an ordinary context free grammar which is unable to capture static semantics' rules concerning typing or scoping, the graphical class diagram notation lacks the expressive power needed to lay down complex constraints that rule out illegal combinations of concepts in the language. Such constraints, known as Well Formedness Rules (WFRs) define the static semantics of a modeling language. They are usually formalized as invariants on the metamodel, using OCL [9] or an OCL dialect.

As they complement the class diagram descriptions, the major value of WFRs resides in the fact that they ensure a better understanding of the modeling concepts' semantics. The existence of an informal (natural language) equivalent of each WFR is mandatory in this respect. Regarding the formal specification, its role is twofold. On the one side, it increases rigor and helps preventing potential ambiguities. On the other side, it lays the basis of automatic model validation. Assuming that the language is tool-supported, the formalized WFRs allow checking whether models are correct/compilable or not with respect to their representation language. No model transformation task, such as code generation or XMI serialization, should be allowed on non-compilable models.

The arguments above are even stronger when the language is a metamodeling language. Metamodeling languages are the "languages used to define other languages". They stand at the top of the metamodeling architectures proposed by all model-driven approaches. In case of a metamodeling language, its abstract syntax is represented by means of a meta-metamodel (a model which is its own metamodel). Having its static semantics appropriately specified (by means of explicit and formal WFRs associated to the meta-metamodel) is thus

[1]Here, we use the term *metamodel* in its general acceptance, as denoting the abstract syntax model. According to the LDD vision however, a metamodel should capture the entire model of a language, covering also its concrete syntax and semantics.

highly important, since it enables checking the compilability of all metamodels instantiating it.

The research carried on the three previously metioned meta-metamodels has allowed us to identify the following general requirements in defining the static semantics of a metamodeling language:

- All WFRs should be stated in both an informal (human-understandable) and formal (machine-readable) style. The informal specification should come prior to the formal one and be as detailed and precise as possible.
- Each informal definition of a WFR should be accompanied by meaningful model test cases, promoting a test-driven WFR specification style.
- The formal specifications should be stated in a manner that allows getting the maximum amount of useful debugging hints in case of assertion failures (specifications should be testing-oriented). The use of OCL specification patterns, as proposed in [4], is highly recommended.
- The OCL specification style should ensure identical WFRs' evaluation results after translation in a programming language.

3. XCore static semantics

XCore is the bootstraping kernel of XMF (eXecutable Metamodeling Facility) [1, 5], a MOF-like metamodeling facility focused on capturing all aspects of a language definition - abstract syntax, concrete syntax and semantics. Unlike MOF though, XMF is completely self-defined and provides platform-independent executability support by means of an executable OCL dialect named XOCL.

3.1. **State of facts.** The official XMF reference [5] acknowledges the value of WFRs and promotes their use in defining the abstract syntax of modeling languages. Still, the document does not describe (neither informally, nor formally) any WFR for the XCore meta-metamodel. As regarding the XMF implementation, this does only include two explicit XOCL constraints, specified in the context of the `Element` and `Object` classes, respectively. Apart from these, there seems to be also a number of other constraints which are only inferable from the XOCL code corresponding to the XCore modifiers.

The XMF approach, that omits the explicit definition of WFRs, trying to preserve model consistency only by means of a suitable implementation of modifiers, has a number of drawbacks.

- In case of an executable language such as XMF, which also provides an interpreter console, one can never assure that model changes will be performed exclusively by calling the corresponding modifiers in the

prescribed order. Direct assignments or different call sequences are also possible, leading to potentially invalid models.
- As emphasized by [6], this approach may be seen as an alternative to the use of preconditions. As opposed to preconditions however, it induces an increased code complexity, with a negative effect on reliability.
- Complex constraints generally involve multiple classes and the necessity of "adjusting" the code of several modifiers. Overlooking to check for the rule in any of these modifiers may lead to incorrect models. Instead, writing explicit WFRs is simpler, more clear, and less error-prone.
- Trying to preserve model consistency at all stable times may not be the best solution always. Underspecification, for instance, may be desirable in particular circumstances.

Writing explicit WFRs is a prerequisite in enforcing them. Even with the approach taken, the XMF implementation does not cover some of the elementary WFRs that are compulsory for object-oriented concepts, such as avoiding name conflincts among features of the same class/classifier or the proper management of contained-container dependencies.

3.2. **Proposed improvements.** As a solution to the above mentioned problems, we have proposed a set of XOCL WFRs for the XCore meta-metamodel, which we have tested on relevant model examples. The entire set of rules, together with the corresponding tests, can be consulted at [2]. Below, we only discuss three relevant examples.

3.2.1. *Name conflicts among owned and inherited members.* As previously stated, one of the WFRs not covered by the XMF implementation concerns the name conflict among an attribute owned by the current class and attributes inherited from its ancestors. This is a fundamental object oriented modeling constraint, being enforced by object oriented programming languages as well. Such a conflict should arise in case of class `D` from Figure 2, which defines the attributes `b` and `r`, having identical names with an inherited attribute and reference, respectively.

The XOCL constraint that we propose for the above mentioned WFR is given below. Figure 1 illustrates the corresponding part of the XCore meametamodel.

[WFR1] There should not be any name conflicts among the attributes owned and inherited by a class.

```
context Attribute @Constraint uniqueName
 let allAtts = self.owner.allAttributes() then
```

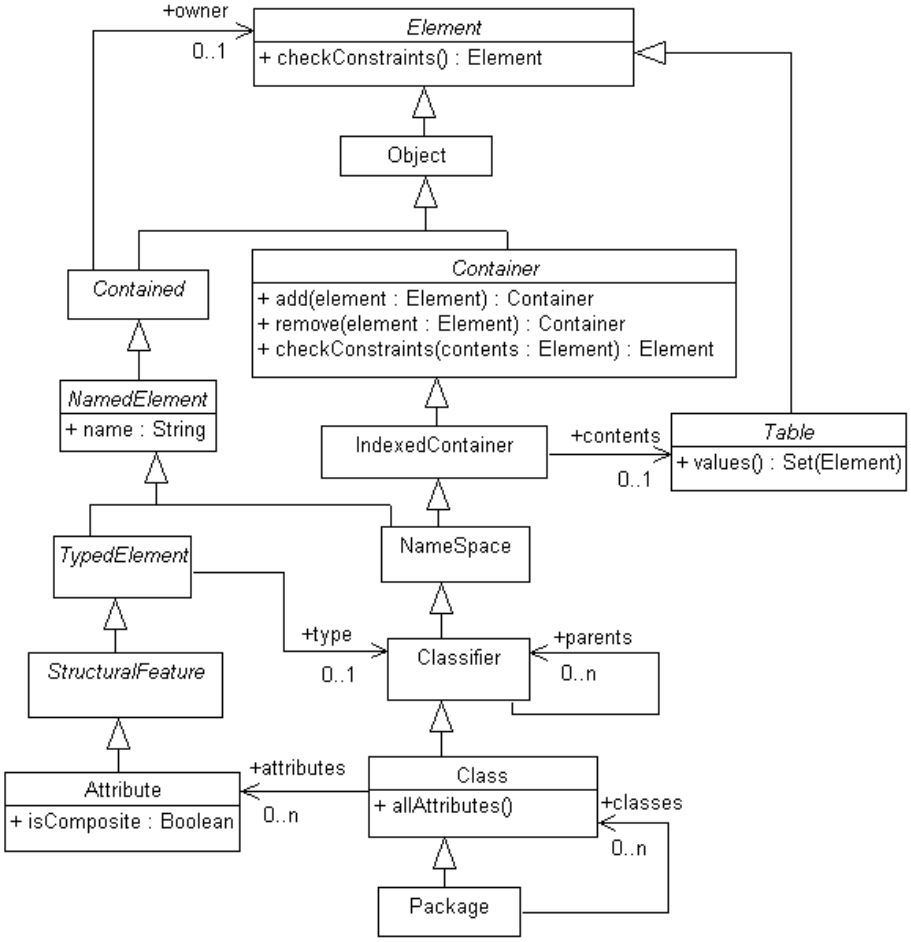

FIGURE 1. An excerpt of the XCore meta-metamodel

```
    sameNameAtts = allAtts->excluding(self)->select(att |
                      att.name.asSymbol() = self.name.asSymbol())
 in sameNameAtts->isEmpty()
 end

fail
  let sameNameAtts = self.owner.allAttributes()->excluding(self)->
         select(att | att.name.asSymbol() = self.name.asSymbol()) then
      msg = "Attribute name duplication! " +
            "Inherited/owned attributes of " + self.owner.toString() +
            " with the same name: "
  in @While not sameNameAtts->isEmpty() do
       let att = sameNameAtts->sel
```

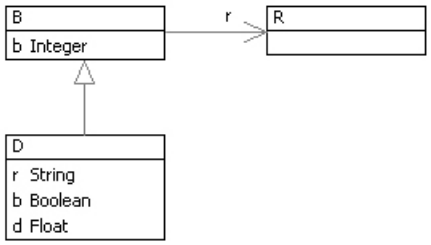

FIGURE 2. A non-valid model example

```
        in msg := msg + att.owner.toString() + "::" + att.toString() + "; ";
           sameNameAtts := sameNameAtts->excluding(att)
        end
      end;
      msg
  end
end
```

Apart from the constraint itself, XMF allows the specification of a fail clause, whose body is intended to provide valuable model debugging information in case of assertion failure. This facility is in accordance to the testing-oriented specification style that we promote in [4].

3.2.2. *Containment relationships.* The proper management of containment (composition) relationships is a fundamental issue in metamodeling. This subject has been also approached by us in [3], in the context of UML 2.3 [10]. As shown by the metamodel excerpt in Figure 1, XCore represents containments explicitely, by providing the `Contained` and `Container` abstract metaclasses in this purpose. Below, we give their description, as taken from the XMF documentation.

> "A *container* has a slot contents that is a table[2]. The table maintains the contained elements indexed by keys. By default the keys for the elements in the table are the elements themselves, but sub-classes of container will modify this feature accordingly. Container provides operations for accessing and managing its contents."
>
> "A *contained* element has an owner. The owner is set when the contained element is added to a container. Removing an owned

[2]According to the metamodel, this rather seems to be the description for `IndexedContainer`, due probably to a lack of synchronization between metamodel and documentation.

element from a container and adding it to another container will change the value of owner in the contained element."

According to the commonly-agreed semantics of containments, we claim that there are two fundamental rules that any model should fulfil in this respect.

(1) A part should belong to a single container at a given time.
(2) A container cannot be itself contained by one of its parts.

As in case of other constraints, the enforcement of the ones above was meant to be covered in XMF by an appropriate implementation of operations in the descendants of `Container` and `Contained`. Moreover, in order to preserve models' validity, these operations are expected to be called in a particular sequence during model editing tasks. As a consequence, the models created using the model/diagram editors of the XMF tool (XMF-Mosaic) are correct with respect to these rules. However, the models edited using the interpreter console (where there is freedom with respect to the type and sequencing of the editing operations) may reach invalid states, which are impossible to detect in the absence of explicitly stated WFRs.

In order to exemplify this for the rule (1), let us start from a sample XCore model containing an empty package named `Test1` (which has been assigned to a global variable `t1`), and the following sequence of XOCL commands executed within the XMF interpreter console.

```
p1 := Package("P1");
t1.add(p1);
p2 := Package("P2");
t1.add(p2);
c := Class("C");
p1.add(c);
```

The lines above modify our initial model by creating two new packages, `P1` and `P2`, which are added as subpackages of `Test1`, and a class, `C`, which is added to package `P1`. As a consequence, class `C` will have `P1` as its `owner`, while `P1` will have `C` as the only element witin its `contents` table.

Suppose `C` has been mistakely added to `P1`, when it should have been, in fact, added to `P2`. Issuing the following command in the console

```
p2.add(c);
```

apparently solves the problem, since the owner of `C` is changed to `P2`, and `C` is added to the contents table of `P2`. However, `C` still belongs to the contents table of `P1`, from which it should have been removed prior to its addition to `P2`. Therefore, in the current state, the model is invalid with respect to rule (1), as `C` simultaneously belongs to two different containers (`P1` and `P2`). A visual proof of this is given by the model browser on the left of the XMF-Mosaic

screenshot from Figure 3, illustrating the state of the model as reached after the execution of the above commands.

Still, even if the model is obviously wrong, the lack of an appropriate WFR makes the call to `checkConstraints()` on `Test1` report this package and its entire contents as valid. The XCore WFR that we propose below offers a solution to this problem.

[WFR2] All Contained instances that belong to the contents table of an IndexedContainer should have that container as owner.

```
context IndexedContainer
 @Constraint validOwnerForContents
  self.contents.values()->select(v | v.oclIsKindOf(Contained) and
              v <> null)->select(v | v.owner <> self)->isEmpty()

  fail "The elements from " +
       self.contents.values()->select(v | v.oclIsKindOf(Contained) and
                    v <> null)->select(v | v.owner <> self).toString() +
       " should have " + self.toString() + " as the owner!"
 end
```

As shown by the right-hand side of the screenshot in Figure 3, the model checking performed after the addition of the above constraint to `IndexedContainer` reports the `P1` package as invalid with respect to this particular constraint. In fact, the proposed constraint captures anomalies of a more general nature than just parts simultaneously belonging to at least two different containers (e.g. parts belonging to the `contents` table of a container and having no `owner` set at all).

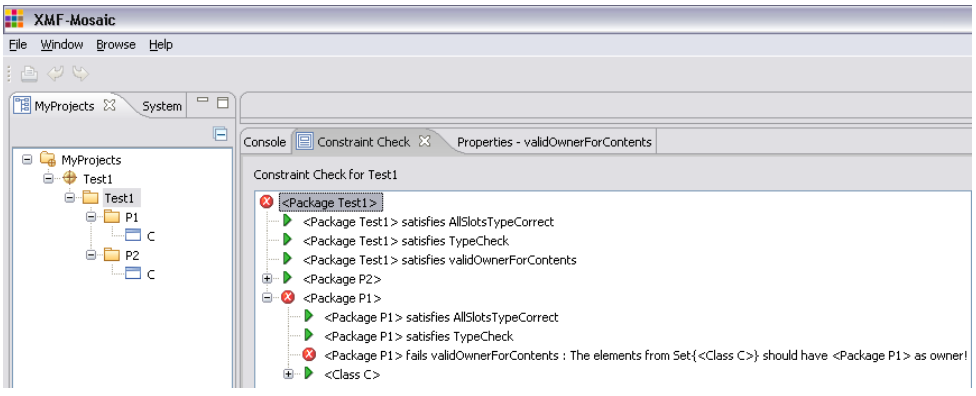

FIGURE 3. XMF-Mosaic screenshot

Regarding the rule (2) above, the neccessity of introducing a corresponding explicit WFR can be argued by means of the following example. Let us

assume the existence of an XCore model consisting of a single empty package named `Test2` (that has been assigned to a global variable `t2`). Furthermore, assume that there is the neccessity of creating under `Test2` a hierarchy of three subpackages, say `P1`, `P2`, and `P3`, each included in the previous one. This basic model editing task can be accomplished in XMF by means of the following sequence of commands.

```
p1 := Package("P1");
t2.add(p1);
p2 := Package("P2");
p1.add(p2);
p3 := Package("P3");
p2.add(p3);
```

However, the misuse of `p1` instead of `p3` as the argument of the latter call above has the effect of creating a circular containment between packages `P1` and `P2`, each of them becoming the `owner` of the other. Yet, in the absence of an explicit WFR prohibiting this, a call to `Element::checkConstraints()` on any of them reports no problem at all.

As a solution to this, we propose the XOCL WFR below, which applies to all indexed containers, except for the `Root` namespace (in XMF, `Root` is the global namespace in which everything is contained, itself included).

[WFR3] No IndexedContainer different from the Root namespace can be owned by one of its parts.

```
context IndexedContainer
 @Constraint notOwnedByPart
  (self <> Root and self.oclIsKindOf(Contained)) implies
  self.contents.values()->select(v | self.owner = v)->isEmpty()

  fail "This container is owned by each of its parts from " +
        self.contents.values()->select(v | self.owner = v).toString()
 end
```

4. Conclusions

In view of the goals pursued by the model-driven approaches, formalizing the static semantics of any metamodeling language is a must. Within this paper, we have analyzed the approach taken in case of the XCore metametamodel and we have identified its shortcomings. The proposed solution consists in the definition of a set of XOCL constraints, that have been validated on relevant model examples. Each WFR is stated both informally and formally and is accompanied by meaningful test cases.

Acknowledgements

This work was supported by CNCSIS-UEFISCSU, project number PNII-IDEI 2049/2008.

References

[1] eXecutable Metamodeling Facility (XMF) homepage. `http://itcentre.tvu.ac.uk/~clark/xmf.html`.

[2] Frame Based on the Extensive Use of Metamodeling for the Specification, Implementation and Validation of Languages and Applications (EMF_SIVLA) homepage. `http://www.cs.ubbcluj.ro/~chiorean/CUEM_SIVLA`.

[3] Dan Chiorean and Vladiela Petraşcu. Specification and Evaluation of Constraints in MOF-based Metamodels. In *ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10), Workshop on OCL and Textual Modeling*, October 2010. (accepted).

[4] Dan Chiorean, Vladiela Petraşcu, and Ileana Ober. Testing-Oriented Improvements of OCL Specification Patterns. In *Proceedings of 2010 IEEE International Conference on Automation, Quality and Testing, Robotics AQTR 2010, Tome II*, pages 143–148. IEEE Computer Society, 2010.

[5] Tony Clark, Paul Sammut, and James Willans. *Applied Metamodeling. A Foundation for Language Driven Development (second edition)*. Ceteva, 2008.

[6] Bertrand Meyer. *Object-Oriented Software Construction (second edition)*. Prentice Hall, 1997.

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

[8] Object Management Group (OMG). Model Driven Architecture (MDA) Guide, Version 1.0.1. `http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf`.

[9] Object Management Group (OMG). Object Constraint Language (OCL), Version 2.2. `http://www.omg.org/spec/OCL/2.2/PDF`.

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

[11] Douglas C. Schmidt. Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[12] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (second edition)*. Addison-Wesley Professional, December 2008.

Babeş-Bolyai University of Cluj-Napoca, Mihail Kogălniceanu nr. 1, Cluj-Napoca, Romania

E-mail address: {vladi,chiorean}@cs.ubbcluj.ro