Embedding Domain-Specific Modelling Languages in Maude Specifications

Vlad Rusu

Embedding Domain-Specific Modelling Languages in Maude Specifications

Vlad Rusu, Inria Lille Nord-Europe & Laboratoire d'Informatique Fondamentale de Lille, France

Vlad.Rusu@inria.fr

Abstract

We propose an approach for embedding Domain-Specific Modelling Languages (DSML) into Maude, based on representing models and metamodels as Maude specifications, and on representing operational semantics and model transformations as computable functions/relations between such specifications. This provides us, on the one hand, with abstract definitions of essential concepts of domain-specific modelling languages: model-to-metamodel conformance, operational semantics, and (operational-semantics-preserving) model transformations; and, on the other hand, with equivalent executable definitions for those concepts, which can be directly used in Maude for formal verification purposes.

1 Introduction

Domain-Specific Modelling Languages (DSML) are modelling languages designed by the people who use them. Typically, the design of a DSML involves the definition of a metamodel describing the language's syntax. Recent works have focused on techniques for defining a language's operational semantics (cf. Related Works, Section 5). Some of these works [1, 2] are based on Maude [3].

We introduce a new approach for embedding DSML in Maude. We represent models and metamodels as Maude specifications, and the operational semantics of DSML, as well as the model transformations between DSML, as computable functions/relations between such specifications. This provides us with abstract definitions for the essential notions of (1) model-to-metamodel conformance, (2) operational semantics, (3) model transformations, and (4) the property of model transformations of being *operational-semantics preserving*. We also obtain equivalent executable definitions for these notions, which can directly be used by Maude.

Contributions. Based on the representation of metamodels and models as Maude specifications proposed in earlier work [4, 5], we obtain a new abstract definition of *model-to-metamodel conformance* as an "inclusion" of the semantics of the model's specification into the semantics of the metamodel's specification. This captures the intuitive idea that a model conforms to a metamodel if the model's semantics is allowed by the metamodel's semantics. Moreover, we show that our proposed abstract definition is equivalent to the Maude executable one, defined in [4, 5], which is used for automatically verifying model-to-metamodel conformance.

Next, we propose abstract definitions for *operational semantics* of DSML and for *model transformations* between DSML, as well as executable definitions for these concepts in Maude. Note that the operational semantics of a DSML is just a particular class of an *endogenous* model transformations, i.e., a transformation between the DSML's metamodel and itself. Thanks to the semantics of metamodels (that we have obtained via the translation of metamodels to Maude specifications), we obtain the abstract definition for model transformations between two DSML as computable *functions/relations between the semantics of their metamodels*. This formalises the intuition that "model transformations are functions/relations between two metamodels". An abstract definition of the operational semantics of a DSML is obtained by considering it as an endogenous transformation of the language's metamodel.

Having obtained abstract definitions for operational semantics and model transformations, what are the equivalent executable definitions in Maude? Such definitions require Maude to "execute" the Maude specifications representing models of our DSML and to transform them according to, e.g., operational semantics rules or to model transformation rules. This is possible thanks to Maude's *reflective* nature: one can define in Maude functions and relations between Maude specifications. We show that the set of computable functions/relations between the semantics of two metamodels coincides with the set of executable functions/relations that can be defined in Maude by reflection in a certain precisely identified way.

A natural question that arises is then: given two DSML $\mathcal{L}_1$ and $\mathcal{L}_2$, each endowed with an operational semantics, and given a model transformation between $\mathcal{L}_1$ and $\mathcal{L}_2$, how to define the fact that the transformation *preserves* the operational semantics of $\mathcal{L}_1$ into $\mathcal{L}_2$? Such transformations are used for verification purposes, e.g., $\mathcal{L}_2$ is the input language of a model checker [**?**]. is An abstract definition capturing this intuition requires the model transformation to be a *simulation* between the *transition systems* underlying the operational semantics of our two DSML defined in Maude. There are many kinds of simulations (see, e.g., [6] for simulations in the context of algebraic specifications). We here propose one that requires the transformation to be a *refinement* of $\mathcal{L}_1$ into $\mathcal{L}_2$, and define a semi-algorithmic procedure in Maude for automatically checking it. This means that if the simulation does not hold (i.e., the model transformation fails to preserve operational semantics) then our procedure will detect this; otherwise, the procedure may not terminate. We also suggest inductive theorem-proving techniques (also available in Maude [7]) for proving that simulation does hold.

Organisation. After this introduction, in Section 2 we briefly present Maude. In Section 3 we present our abstract and equivalent Maude-executable definitions of essential notions related to DSML: metamodel, model, conformance, operational semantics, and model transformations. In Section 4 we study model trans-

```
spec ELEMENT-SET is
sorts Element Set
subsort Element <  Set
a b :  Element
empty : Set
_,_ : Set Set ->  Set [assoc comm id:empty]
X:Element, X:Element = X:Element
```

Figure 1: MEL specification `ELEMENT-SET`.

formations that preserve operational semantics. Section 5 presents related and future work, and concludes.

2 Background

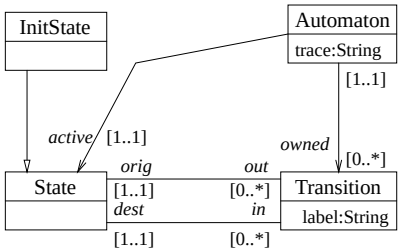
Maude specifications are written in Membership Equational Logic (MEL) or Rewriting Logic (RL), a superset of MEL. We present them here by means of examples. The reference for Maude is [3].

A MEL specification consists of a set of *sorts*; of a partial order on sorts called the *subsorting* relation, which expresses the fact that some sorts are subsorts of others; of a set of *operations*, which are functions between the sorts, each of which has an *arity* giving its number of arguments, where constants are 0-ary functions; and of a set of *axioms* defining the operations. Axioms are (possibly conditional) *equations* between terms, or *memberships* of terms into sorts. Among the equational axioms, some particularly important ones (associativity, commutativity, identity, ...) can be associated to some operators, saving us the trouble of writing an explicit equation. A *term* is either a constant, a variable of a given sort, or the application of an operation to the appropriate number of terms of the appropriate sorts. A *ground term* is a term without variables.

Rewriting Logic is a superset of MEL, which allows, in addition to all the above, for (possibly conditional) *rewrite rules*.

A sample MEL specification is shown in Figure 1. It is the standard way of defining (finite) *sets* in Maude. We use a simplified Maude syntax for better readability. Sets are constructed using the `empty` constant, or by taking *unions* of sets, denoted by the `_,_` operation in Figure 1, which is declared to be associative, commutative, and to have `empty` as its identity element. There is a sort `Element` for elements, which consists of the constants `a` and `b`, and the subsorting relation `Element < Set`, which says that every element is a set. Note that, with just this definition, a set would allow for multiple identical elements. To avoid this, the equation `X:Element , X:Element = X:Element` prevents elements to occur in a set more than once. However, if this equation was replaced by a *rewrite rule*, written in Maude-like syntax as `X:Element , X:Element => X:Element`, the intepretation would be different: the equation is a part of the definition of sets; by contrast, the rule can be part of the definition of the *operational semantics* of a system whose states are multisets.

The *semantics* of a MEL specification is defined in terms of *algebras*. Defining an algebra for a specification S consists in interpreting each sort of S as a set such that the subsorting relation is interpreted by the subset relation. The operations are then interpreted as functions between the corresponding sets (or by constants in the corresponding sets). It is required that the interpretation satisfies the specification's axioms. We denote by $\mathcal{A} \models \phi$ the satisfaction of a formula ϕ of a specification S by an algebra $\mathcal{A}$ of S, with the usual meaning - when interpreted in $\mathcal{A}$, ϕ evaluates to *true*.



Context $State : self.out \rightarrow forAll(t_1, t_2 | t_1 \neq t_2 \; implies \; t_1.label \neq t_2.label)$
Context $Transition : self.label \neq ""$)

Figure 2: Metamodel for deterministic finite automata.

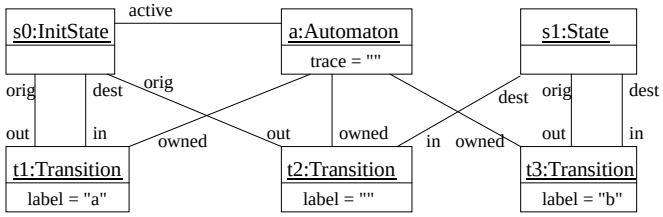


Figure 3: Model of an automaton.

The *initial algebra* of a MEL specification is intuitively the "most natural interpretation" of the specification; for the specification depicted in Figure 1 it consists of sets of `a`s and `b`s. Formally, the initial algebra interprets each sort s as the *set of equivalence classes of ground terms* that can be proved to be of sort s using MEL's deductive system [8] - where two terms are in the same equivalence class iff they can be proved equal using that same deductive system. The functions interpreting the non-constant operations are then implicitly defined by the specifications's axioms.

The *initial semantics* of a MEL specification consists of its initial algebra. We denote $(\!|S|\!)$ the initial semantics of a specification S.

The *loose semantics* of a MEL specification S, denoted by $[\![S]\!]$, is the set of all its algebras. We use the initial semantics for the MEL specifications denoting models, and a finitary version of the loose semantics for the MEL specifications representing metamodels.

The semantics of a Maude RL specification is any *transition system* whose states interpret terms, and whose transition relation interprets the *rewrite relation* of the RL specification (two terms t_1, t_2 are in relation if t_2 is obtained from t_1 by one rewrite).

Finally, in order to make the definitions of operational semantics and of model transformations *executable* by Maude, we shall use the fact that Maude is *reflective*: there exists a Maude specification that reflects all Maude specifications, including itself. The specification in Figure 1 is obtained by applying a certain operation to the following parameters: a set of sorts (here, `Element` and `Set`), a subsorting relation (here, `Element < Set`), a set of operations (here, `a, b, empty`, and `_,_`) and a set of axioms (here, the sole equation `X:Element , X:Element = X:Element`).

3 Representing DSML into Maude

In this section we propose abstract definitions to the essential notions involved in DSML: metamodel, model, model-to-metamodel conformance, operational semantics, and model transformations. For conformance, operational semantics, and model transformations we show the equivalence of abstract definitions with operational definitions that can be used by Maude for verification.

We take in this paper the commonly shared view that metamodels are essentially UML class diagrams. An example is depicted in Figure 2. It is meant to capture the usual finite automata.

The unidirectional association from the class *Automaton* to the class *State* denotes the *active* state. The *InitState* subclass of *State* represents initial states of automata. The class *Automaton* has the *trace* attribute - a string of characters, obtained by concatenating *labels* of *transitions* fired by the automaton. Transitions are associated with *orig*in and *dest*ination states. The opposite roles, from the point of view of states, are those of *in*coming and *out*going transitions. The roles of associations are labelled with multplicities, e.g., transitions have one origin and one destination state. The OCL *invariants* below the diagram say that the automaton is deterministic: for each distinct pair of transitions originating from the same state, their labels are different, and all labels are nonempty.

Figure 3 shows a model of an automaton as an UML object diagram of the class diagram in Figure 2. It is composed of: a self-loop labelled "a" on the (active and initial) state s_0; a transition from state s_0 to state s_1 labelled ""; and a self-loop labelled "b" on s_1. This model does not conform to the metamodel in Figure 2 because it violates the metamodel's second OCL invariant.

Model and metamodel representations in Maude. We give semantics to (meta)models by representing them in Maude.

We first discuss the existing alternatives. Existing approaches [1, 2] represent models as Maude terms. However, [1] represents metamodels as sorts (specifying the constraints that models conforming to a given metamodel must satisfy), and [2] base their representation on Maude's object oriented extension. In our opinion both approaches [1, 2] miss the opportunity of using the *existing rich semantics* provided by *standard order-sorted specifications* (technically, MEL specifications without explicit membership axioms), which (as we will see) provides us with relatively simple and natural abstract definitions to model and metamodel semantics, to conformance, as well as to operational semantics and to model transformations. By doing so, we avoid the complexity of expressing conformance with memberships, or having to rely on Maude's object-oriented extension.

Moreover, both approaches [1, 2] suffer from a theoretical problem: the sort definitions (respectively, the object-oriented specifications) denoting metamodels are quite complex, with the consequence that (to our best knowledge) their representations of model-to-metamodel conformance were not shown to be decidable.

Hence, we take a different approach - we represent both metamodels and models as order-sorted specifications. The constructions present in metamodels: classes, inheritance between classes, associations, and attributes of classes, are mapped to individual constructions of order-sorted specifications: respectively, to sorts, to subsorting relations, and to functions between sorts. Constructions present in models are also mapped to corresponding constructions of order-sorted specifications, and OCL invariants are mapped to equations, such that, overall, the specifications representing object diagrams are *ground confluent and terminating* [4]. This ensures (and in Maude, is required for) the decidability of model-to-metamodel conformance, especially in the (usual) case when metamodels include OCL invariants. This is an advantage with respect to the above-mentioned approaches, and motivates our choice representing (meta)-models as order-sorted specifications.

3.1 Metamodels

We denote by $\text{MEL}(\mathcal{MM})$ the translation of a metamodel $\mathcal{MM}$ to MEL (actually, to the order-sorted subset of MEL), defined by:

- the standard specifications of elementary types occuring in the metamodel (e.g., Boolean, Integer, String, ...), are imported in $\text{MEL}(\mathcal{MM})$,
- each class c is translated into a sort c. The generic sort *Set* of Maude is instantiated to c and the result $Set\{c\}$ is also imported in $\text{MEL}(\mathcal{MM})$,
- the inheritance relation is represented by the subsorting relation: whenever c_1 directly inherits from c_2 in $\mathcal{MM}$ we have in $\text{MEL}(\mathcal{MM})$ a declaration $c_1 < c_2$;
- each attribute a of type t of a class c is translated to a function declaration $a : c \rightarrow t$;
- each uni-directional association between classes c_1 and c_2, where c_2 plays the role r_2, is translated into a function $r_2 : c_1 \rightarrow Set\{c_2\}$;
- a bidirectional association is translated as two uni-directional associations;
- if the metamodel contains OCL invariants they are translated as defined in [4].

We do not give details about the translation [4] of OCL invariants due to lack of space; how exactly the translation is performed is irrelevant here - it suffices for us to know that the translation exists and that it generates a finite set of ground confluent equations.

Other features of metamodels (roles with multiplicities, composition and aggregation associations...) are not translated since they do not any expressiveness to metamodels and can be equivalently encoded in OCL. For the metamodel shown in Figure 2, the result of the translation is shown in Figure 4, using a simplified Maude syntax. The last two statements are ad-hoc translations of the OCL invariants of the metamodel (the actual representation [4] is syntactically more complex but semantically equivalent). Other OCL invariants, also translated as equations (not shown in the figure encode multiplicity constraints of the association roles. For example, the equation `card(active(a:Automaton)) = 1` encodes the `1..1` multiplicity on the `active` role (Figure 2).

For a MEL specification S we denote by $[\![S]\!]_f$ the set of algebras $\mathcal{A}$ of S such that

- the restriction of $\mathcal{A}$ to the specifications imported in S is their initial algebra, and
- $\mathcal{A}$ interprets each proper sort of S (i.e., a sort that is not imported) as a finite set.

We shall call $[\![S]\!]_f$ the *finitely loose semantics of* S.

Definition 1 (metamodel semantics) *The semantics of a metamodel $\mathcal{MM}$ is the finitely loose semantics $[\![\text{MEL}(\mathcal{MM})]\!]_f$ of the* MEL *representation of $\mathcal{MM}$.*

```
spec Deterministic-Automata-Meta-Model is
import Bool String
sorts Automaton Transition State InitialState
subsort InitialState < State
owned : Automaton -> Set{Transition}
orig : Transition -> Set{State}
dest : Transition -> Set{State}
incoming : State -> Set{Transition}
outgoing : State -> Set{Transition}
trace : Automaton -> String
active : Automaton -> State
label : Transition -> String
equals(label(t1:Transition), label(t2:Transition))  = false
  if outgoing(x:State) :=
  t1:Transition, t2:Transition, S:Set{Transition}
length(label(x:Transition))  > 0  = true
```

Figure 4: MEL specification of the metamodel in Fig. 2.

```
spec Automaton-Model is
... import DeterministicAutomata-Meta-Model except for
... the equations that denote OCL invariants
a : Automaton
s0   : InitialState
s1 : State
t1 t2 t3   :  Transition
owned(a) = t1, t2, t3 .
trace(a) = ""
active(a) = s0
orig(t1) = s0
dest(t1) = s0
label(t1) = "a"
orig(t2) = s0
dest(t2) = s1
label(t2) = ""
orig(t3) = s1
dest(t3) = s1
label(t3) = "c"
incoming(s0) = t1
outgoing(s0) = t1, t2
incoming(s1) = t1, t3
outgoing(s1) = t3
```

Figure 5: MEL specification of the model depicted in Figure 3.

The reason why we use the finitely loose semantics (instead of the "plain" loose semantics $[\![\text{MEL}(\mathcal{MM})]\!]$) is that the instances of a metamodel (i.e., its models) are finite, whereas an algebra of a specification may be infinite. Also, this avoids undesired changes to the semantics of the elementary types (Booleans, Strings, ...), hence the initial-algebra requirement on those types.

3.2 Models

We now describe the translation of models $\mathcal{M}$ to MEL (order-sorted) specifications. A model $\mathcal{M}$ is essentially an object diagram of some metamodel (i.e., class diagram) $\mathcal{MM}$[1]. If this is the case we simply say that $\mathcal{M}$ *is based on* $\mathcal{MM}$. We shall denote by $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ the MEL specification defined as follows. First, the MEL translation of the metamodel $\mathcal{MM}$ (with the exception of the axioms translating the metamodel's OCL invariants - for technical reasons) is generated and imported in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$. Then,

1. each instance o of class c becomes a declaration $o :\rightarrow c$ of a constant o of sort c;

2. each attribute a of an object o having value v is translated to an equation $a(o) = v$;

3. the set of links of an association between an object o and a set of objects O, each playing the role r from the point of view of o, is translated as an equation $r(o) = O$.

For the model $\mathcal{M}$ depicted in Figure 3 and the metamodel $\mathcal{MM}$ depicted in Figure 2, the MEL specification $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ is depicted in Figure 5.

Definition 2 (model semantics) *The semantics of model $\mathcal{M}$ based on metamodel $\mathcal{MM}$ is $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!)$.*

Definition 3 (conformance) *$\mathcal{M}$ conforms to $\mathcal{MM}$ if $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \in [\![\text{MEL}(\mathcal{MM})]\!]_f$.*

Note that the above definition implicitly says that $\mathcal{M}$ is based on $\mathcal{MM}$ (otherwise, $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ is not defined). We now show that Definition 3 of conformance is equivalent to our operational definition of conformance from [5]. There, conformance is defined as follows. For a model $\mathcal{M}$ based on a metamodel $\mathcal{MM}$, the equational representation of the conjunction of all OCL invariants of $\mathcal{MM}$, denoted here by $\text{OCL}_{\text{MEL}}(\mathcal{MM})$, is automatically evaluated in the Maude representation $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ by equational reduction, thanks to the ground confluence and termination of the equations denoting OCL invariants [4]. Conformance holds iff the canonical form of the conjunction $\text{OCL}_{\text{MEL}}(\mathcal{MM})$ in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ is *true*. Since for ground confluent and terminating (order-sorted) MEL specifications, the initial algebra is the algebra of canonical forms of terms [3], we obtain that conformance in the sense of [5] amounts to the satisfaction relation $\boxed{(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})}$. We need the following:

Lemma 1 (metamodel semantics=set of semantics of its models) $[\![\text{MEL}(\mathcal{MM})]\!]_f = \{(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$.

Proof (sketch): To prove the $\subseteq$ inclusion, from any algebra $\mathcal{A} \in [\![\text{MEL}(\mathcal{MM})]\!]_f$ we build a specification $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ and show that its initial algebra is the algebra $\mathcal{A}$ we started from. For this, we take the (finite) sets in the algebra $\mathcal{A}$ interpreting the sorts of $\text{MEL}(\mathcal{MM})$ denoting classes of $\mathcal{MM}$, and declare the elements of those sets as constants of the respective sorts in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$; and characterise the non-constant functions in the algebra $\mathcal{A}$ with equations in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$. The initial algebra $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!)$ then coincides with the algebra $\mathcal{A}$ we started from. To conclude, note that $\mathcal{A} \in [\![\text{MEL}(\mathcal{MM})]\!]_f$ just means $\mathcal{A} \models \text{OCL}_{\text{MEL}}(\mathcal{MM})$.
$\supseteq$: consider a model $\mathcal{M}$ based on $\mathcal{MM}$ such that $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})$. The initial algebra $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!)$ is then an algebra in $[\![\text{MEL}(\mathcal{MM})]\!]_f$ - the interpretations of sorts of $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!)$ denoting classes of $\mathcal{MM}$ are indeed finite, namely, they consist of the finitely many constants declared in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$; and $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})$ means satisfation of the invariants of $\mathcal{MM}$. □

Lemma 1 implies that the conformance of $\mathcal{M}$ to $\mathcal{MM}$ according to Definition 3: $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \in [\![\text{MEL}(\mathcal{MM})]\!]_f$ is equivalent to $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \in \{(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$, i.e., to $\boxed{(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})}$ that, as seen earlier, is equivalent to conformance in the operational sense given in [5]: thus, executable and abstract definitions of model-to-metamodel conformance coincide.

[1] An object diagram is *of* a class diagram if all objects have classes that belong to the class diagram; all atributes of an object are present in the object's class, and the value of the attributes have the same types as (or have subtypes of) the types declared in the class; and for all links between two objects, there exists an instance between the two object's classes in the class diagram. For technical reasons we assume that all attributes of all objects have values.

We also have the following technical lemma, used later in this section for establishing equivalences between abstract and executable definitions of operational semantics/model transormations.

Lemma 2 *There is a bijection between the sets* $\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ *and* $\{(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$.

Proof (sketch): consider the mapping that to each specification associates its initial algebra. It is obviously a surjection between our two sets. To prove its injectiveness, we note that different models $\mathcal{M}_1$, $\mathcal{M}_2$ of $\mathcal{MM}$ have at least two distinct objects, or different values for the same attribute of an object, or different links between objects. Hence, the respective specifications $\text{MEL}_{\mathcal{MM}}(\mathcal{M}_1)$, $\text{MEL}_{\mathcal{MM}}(\mathcal{M}_2)$ differ either in their constant declarations or in their equation sets (or both). Since by construction there are no equations in specifications of the form $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ between the constants denoting objects, the initial algebra of $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$ interprets sorts as the constants defined of the respective sorts in $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$, and interprets the functions between the sort interpretations as defined by the equations of $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$. Hence, for different models $\mathcal{M}_1$, $\mathcal{M}_2$ of $\mathcal{MM}$, either the sort intepretations or the functions intepretations (or both) differ, hence, we obtain $(\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M}_1)|\!) \neq (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M}_2)|\!)$. □

3.3 Operational Semantics

The operational semantics of a DSML is, intuitively, a function that maps models in the DSML to "next" models. Using the semantics of metamodels (Definition 1) we obtain the following abstract definition for the operational semantics of a DSML, capturing the intuition that a model may have several (finitely many) successors, or none; and that the set of successors should be computable.

Definition 4 (operational semantics) *The operational semantics of a* DSML *of metamodel* $\mathcal{MM}$ *is a recursive function* $F : [\![\mathcal{MM}]\!]_f \rightarrow \mathcal{P}_f([\![\mathcal{MM}]\!]_f)$.

Here, $\mathcal{P}_f(S)$ denotes the set of finite subsets of S. We prove that this abstract definition is equivalent to a Maude-executable definition that can be used for verification purposes; we shall illustrate this on a simple example base on the automata models/metamodel.

Given the bijection between the semantics $[\![\text{MEL}(\mathcal{MM})]\!]_f$ of a DSML's metamodel and the set $\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ (cf. Lemmas 1, 2), we can state the following equivalent definition to Definition 4: the operational semantics of a DSML of metamodel $\mathcal{MM}$ is a recursive function from the set $\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ to the set $\mathcal{P}_f(\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\})$. Moreover, these sets can be algebraically specificied in Maude using the fact that Maude is *reflective*: there exists a MEL specification called `Meta-Module` where all MEL specifications (including itself) are reflected as *terms* of a certain sort called `Module`. We then write in Maude a specification $\texttt{ModelsIn}\mathcal{MM}$ extending `Meta-Module`, where we define a subsort $ModelsIn\mathcal{MM}$ of `Module`, which is interpreted as the set $\{\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})} \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ in the initial algebra of the specification $\texttt{ModelsIn}\mathcal{MM}$ - here, $\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})}$ is the *term* in the specification $\texttt{ModelsIn}\mathcal{MM}$ that reflects $\text{MEL}_{\mathcal{MM}}(\mathcal{M})$. The sort $ModelsIn\mathcal{MM}$ is defined using a conditional *membership*, whose condition checks that our conformance-checking procedure from [5] returns `true`[2]. A definition of a sort of the form $ModelsIn\mathcal{MM}$ is in Figure 7.

Injectiveness of reflection ensures that the sets $\{\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})} \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ and $\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ are in bijection, and the latter is in bijection with $[\![\text{MEL}(\mathcal{MM})]\!]_f$ (cf. Lemmas 1, 2). Hence, an equivalent to Definition 4 is: the operational semantics of a DSML of metamodel $\mathcal{MM}$ is any recursive function in the set $\{\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})} \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\} \rightarrow \mathcal{P}_f(\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\})$. Next, a theorem by Bergstra and Tucker [9] says that recursive functions on a given domain/codomain are exactly those functions that can be defined by a set of ground confluent and terminating equations on algebraic specifications of the domain/codomain. But we have seen that the domain $\{\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})} \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ is algebraically specified as the sort $ModelsIn\mathcal{MM}$ in a certain MEL specification $\texttt{ModelsIn}\mathcal{MM}$; and the polymorphic sort $Set\{\cdot\}$ faithfully encodes finite sets of its argument sort. Hence, the recursive functions from $\{\overline{\text{MEL}_{\mathcal{MM}}(\mathcal{M})} \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\}$ to $\mathcal{P}_f(\{\text{MEL}_{\mathcal{MM}}(\mathcal{M}) \mid (\!|\text{MEL}_{\mathcal{MM}}(\mathcal{M})|\!) \models \text{OCL}_{\text{MEL}}(\mathcal{MM})\})$ are exactly the functions from $ModelsIn\mathcal{MM}$ to $Set\{ModelsIn\mathcal{MM}\}$ that can be equationally defined in some MEL specification containing the specification $\texttt{ModelsIn}\mathcal{MM}$.

Definition 5 (operational semantics (bis)) *The operational semantics of a* DSML *of metamodel* $\mathcal{MM}$ *is any function* $F : ModelsIn\mathcal{MM} \rightarrow Set\{ModelsIn\mathcal{MM}\}$ *that can be equationally defined in a* MEL *specification containing* $\texttt{ModelsIn}\mathcal{MM}$.

This definition is an executable one, in the sense that it refers to equationally-defined Maude functions. However, in order to use Maude's automatic verification tools (namely, state-space exploration, an example is given below) it is better to equivalently represent such nondeterministic semantics using *rewrite rules* of RL.

This can always be done thanks to the following observations. On the one hand, the graph of any function $F : ModelsIn\mathcal{MM} \rightarrow Set\{ModelsIn\mathcal{MM}\}$ can be encoded as the rewrite relation generated by the rewrite rule $x \Rightarrow y \ if \ y, z := F(x)$, where x and y are variables of sort $ModelsIn\mathcal{MM}$, and the variable z has sort $Set\{ModelsIn\mathcal{MM}\}$. On the other hand, the transition relation over the sort $ModelsIn\mathcal{MM}$ of any RL specification containing the MEL specification $\texttt{ModelsIn}\mathcal{MM}$ is a computable, i.e., recursive function from $ModelsIn\mathcal{MM}$ to $Set\{ModelsIn\mathcal{MM}\}$.

Definition 6 (operational semantics(ter)) *The operational semantics of a* DSML *of metamodel* $\mathcal{MM}$ *is the rewrite relation over the sort* $ModelsIn\mathcal{MM}$*, of some rewriting-logic specification containing the specification* $\texttt{ModelsIn}\mathcal{MM}$.

[2]The condition implicitly checks that $\mathcal{M}$ is a valid model based on $\mathcal{MM}$, i.e., that it has no dangling edges, and that it only uses sorts and operations declared in $\text{MEL}(\mathcal{MM})$; these checks are performed by Maude's parser and typechecker.

[3]Note the analogy with [1], where metamodels are encoded as sorts and models are encoded as terms of those sorts. The difference is that [1] perform their encoding directly in Maude's logic, whereas, in our case, Maude's reflection mechanism automatically reflects models as terms/metamodels-as-sorts, based on our encoding of models/metamodels as specifications.

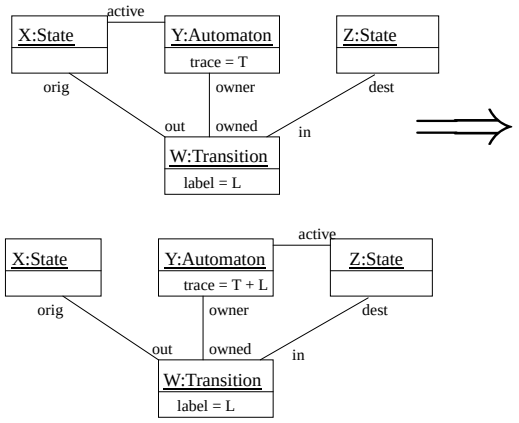


Figure 6: Executing automata: graphical rule.

```
spec Execution is
import Meta-Module Automata-Meta-Model
sort  ModelsInAutomata-Meta-Model
subsort ModelsInAutomata-Meta-Model < Module
--- definition of the sort ModelsInAutomata-Meta-Model
var X : Module
X : ModelsInAutomata-Meta-Model
 if  conformance-check(X,'Automata-Meta-Model) = true
--- rule fort executing automata
('active[Y:Term]  = X:Term,
'owned[Y:Term] = W:Term,
'orig[W:Term]  =  X:Term,
'dest[W:Term]  =  Z:Term,
'label[W:Term]  =  L:Term,
'trace[Y:Term]  =  T:Term)
=>
('active[Y:Term]  = Z:Term,
'owned[Y:Term] = W:Term,
'orig[W:Term]  =  X:Term,
'dest[W:Term]  =  Z:Term,
'label[W:Term]  =  L:Term,
'trace[Y:Term]  =  '_+_[L:Term, T:Term] ) .
```

Figure 7: Executing automata: Maude rewrite rule.

We illustrate below the executable definitions on the metamodel for automata, represented in Maude in Figure 4. For more generality we allow for nondeterministic automata and semantics, hence, we disregard the OCL invariants of that metamodel that encode determinism; we let also let `Automata-Meta-Model` be a copy of the Maude specification in Figure 4 without the last two equations denoting the OCL invariants that encode determinism.

Figure 6 depicts automata execution: if an automaton Y owns a transition W with label L whose origin is X and destination is Z, and the currently active state is X, then the active state becomes Z, and the label L is concatenated to the automaton's trace T.

The corresponding Maude rewrite rule is shown in Figure 7. It closely matches the graphical rule: the links and attribute values are denoted by equations; the rule changes the set of equations in order to change the links and attribute values. Here, the link that changes is the *active* link, from `'active[Y:Term] = X:Term` to `'active[Y:Term] = Z:Term`. Operator names from the meta-model specification are quoted, and variables are of sort `Term`; this is due to the fact that we are using Maude's reflection (allowed by the importation of the `Meta-Module` Maude specification). The attribute value that changes is `'trace[Y:Term]`, whose next-state value is the concatenation of the label L and trace T, expressed by reflection in Maude by the equation `'trace[Y:Term] = '_+_[L:Term, T:Term]`.

We can use now the Maude specification shown in Figure 7 to execute, e.g., the automaton whose model's specification in Maude is shown in Figure 5 and to verify its properties. For example, the following command asks Maude whether an execution of the automaton exist such that the trace of the automaton is "aaabb":
`search in Execution : upModule('Automaton-Model) ⇒* X such that getTrace(X) = '"aaabb".String`.

Maude instantly responds positively, and provides us upon request with the shortest path leading to the solution.

3.4 Model Transformations

The operational semantics of DSML as defined in the previous section is just a particular case of an *endogenous* model transformation, i.e., a transformation where the source and target metamodels are the same. We naturally extend the abstract Definition 4 to model transformations between two different metamodels $\mathcal{MM}_1$ and $\mathcal{MM}_2$, as functions with domain $[\![\mathcal{MM}_1]\!]_f$ and co-domain $\mathcal{P}_f([\![\mathcal{MM}_2]\!]_f)$. We also extend the executable Definitions 5, 6 to model transformations, based on sorts $ModelsIn\mathcal{MM}_1$ and $\mathcal{P}_f(ModelsIn\mathcal{MM}_2)$ defined by reflection as in Section 3.3.

4 Semantical Preservation

Given two DSML $\mathcal{L}_1$ and $\mathcal{L}_2$, each endowed with an operational semantics, and given a model transformation between $\mathcal{L}_1$ and $\mathcal{L}_2$, how to define the fact that the transformation *preserves* the operational semantics when translating from $\mathcal{L}_1$ to $\mathcal{L}_2$? Intuitively, this preservation means that the image in $\mathcal{L}_2$ of any model in $\mathcal{L}_1$ by the transformation does "at least as much" as the original.

In this section we define a notion of semantics-preserving model transformation (with respect to a given notion of simulation), and propose an automatic procedure to detect that the semantics-preservation property does not hold. The procedure is complete: it detects all situations where simulation does not hold, and may not terminate otherwise. The procedure also makes it possible to prove that simulation does hold, using Maude's theorem prover [7].

We naturally identify a meta-model $\mathcal{MM}$ with the set of models that conform to it, and the operational semantics $\rightarrow$ of a DSML having metamodel $\mathcal{MM}$ with a relation $\rightarrow \subseteq \mathcal{MM} \times \mathcal{MM}$. By choosing an "initial state" $\mathcal{M}^0 \in \mathcal{MM}$ we obtain a transition system $\langle \mathcal{MM}, \mathcal{M}^0, \rightarrow \rangle$, which expresses the evolution of models conforming to $\mathcal{MM}$ starting from $\mathcal{M}^0$ and according to $\rightarrow$. We require that such transition systems $\langle \mathcal{MM}, \mathcal{M}^0, \rightarrow \rangle$ to be *non blocking*, meaning that a model can always evolve into some model (possibly, itself). Some definitions for transition systems follow.

For any transition system $\mathcal{T} = (A, a_{ini}, \rightarrow_{\mathcal{A}})$, an execution is a sequence of states $\rho = a_0, \ldots a_n \in A$, such that $a_i \rightarrow_{\mathcal{A}} a_{i+1}$ for $i = 0, \ldots, n-1$; $length(\rho) = n$ is the *length* of the execution ρ. Executions of length 0 are states. We denote by $exec(\mathcal{T})$ the subset of executions that start in a_{ini}.

Definition 7 *Given two transition systems $\mathcal{T} = (A, a_{ini}, \rightarrow_{\mathcal{T}})$, $\mathcal{T}' = (B, b_{ini}, \rightarrow_{\mathcal{T}'})$, a relation $R \subseteq A \times B$, and executions $\rho \in exec(\mathcal{T})$ and $\pi \in exec(\mathcal{T}')$, we say that ρ is* R-matched *by π if there exists $\alpha : [0, \ldots, length(\rho)] \rightarrow \mathbb{N}$ with $\alpha(0) = 0$, such that for all $i \in [0, \ldots, length(\rho)]$, $(\rho(i), \pi(\alpha(i))) \in R$, and such that for all $i \in [0, \ldots, length(\rho) - 1]$, $\alpha(i+1) \in \{\alpha(i) + 1, \alpha(i)\}$.*

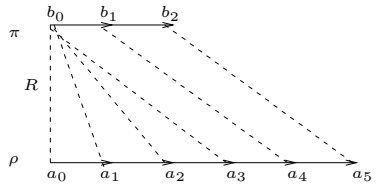


Figure 8: R-matching executions. R is depicted by dashed lines.

In Figure 8 we represent two executions ρ and π. The relation R is denoted by the dashed lines. The function $\alpha : [0, \ldots, 5] \rightarrow \mathbb{N}$ defined by $\alpha(0..3) = 0$ and $\alpha(4) = 1$, $\alpha(5) = 2$ ensures that ρ (of length 5) is R-matched by π (of length 2).

The notion of R-matching allows shorter executions π to match longer executions ρ. This is useful when semantics have different granularities; for example, in a determinisation model-transformation, each execution of possibly nondeterministic automaton is matched (in terms of its trace) by some typically shorter execution of its deterministic automaton. Executions in nondeterministic automata are longer than the executions that match them in deterministic automata because of ""-labeled transitions.

Definition 8 *For transition systems $\mathcal{T} = (A, a_{ini}, \rightarrow_{\mathcal{T}})$, $\mathcal{T}' = (B, b_{ini}, \rightarrow_{\mathcal{T}'})$ and relation $R \subseteq A \times B$, we say that R is a* simulation *between $\mathcal{T}$ and $\mathcal{T}'$ if for all executions $\rho \in exec(\mathcal{T})$ there exists an execution $\pi \in exec(\mathcal{T}')$ s.t. ρ is R-matched by π.*

We are now ready to define semantical preservation. Assume two DSML $\mathcal{L}$ and $\mathcal{L}'$ whose metamodels are $\mathcal{MM}, \mathcal{MM}'$, whose operational semantics give rise to respective transition relations $\rightarrow \subseteq \mathcal{MM} \times \mathcal{MM}$ and $\rightarrow' \subseteq \mathcal{MM}' \times \mathcal{MM}'$, and whose initial states are $\mathcal{M}_0$ and $\mathcal{M}'_0$. Assume also a model tranformation φ between $\mathcal{MM}$ and $\mathcal{MM}'$, i.e., a relation $\varphi \subseteq \mathcal{MM} \times \mathcal{MM}'$.

Definition 9 (semantics-preserving model transformation) *A model transformation φ is* semantics-preserving *if it is a simulation between $\langle \mathcal{MM}, \mathcal{M}^0, \rightarrow \rangle$ and $\langle \mathcal{MM}', \mathcal{M}'^0, \rightarrow' \rangle$.*

To check semantical preservation in Maude, we write two functions `run1step` and `run1step'`, which take a set of models in $\mathcal{MM}$ and in $\mathcal{MM}'$, respectively, and apply one step of the operational semantics of $\mathcal{MM}$ and of $\mathcal{MM}'$, respectively. We then write a conditional rewrite rule:

(†) $\langle \mathcal{M}, \mathcal{S} \rangle \Rightarrow \langle \mathcal{M}', \mathcal{S}' \rangle$
`if` $\mathcal{M}', \mathcal{S}''$ `:=` `run1step`$(\mathcal{M})$ $\wedge$
$\mathcal{S}'$ `:=` $(\mathcal{S}$`,run1step'`$(\mathcal{S}))$ $\cap$ $\varphi(\mathcal{M}')$

That is, any pair $\langle \mathcal{M}, \mathcal{S} \rangle$ is rewritten to some pair $\langle \mathcal{M}', \mathcal{S}' \rangle$ where

- $\mathcal{M}'$ is *some* 1-step successor of $\mathcal{M}$ according to the operational semantics of $\mathcal{MM}$,
- $\mathcal{S}'$ is the intersection between $\varphi(\mathcal{M}')$ and the *union* (denoted by `_,_`) between $\mathcal{S}$ and `run1step'`$(\mathcal{S})$.

Our procedure consists in performing the Maude command:

(‡) `search` $\langle \mathcal{M}_0, \mathcal{M}'_0 \rangle$ `=>*` $\langle \mathcal{M}, \emptyset \rangle$.

Proposition 1 *A model transformation φ is semantics-preserving if and only if the Maude search command (‡) finds no solution.*

Proof (sketch): based on the following statement, easily established by induction. For each pair of the form $\langle \mathcal{M}, \mathcal{S} \rangle$ reachable in n rewriting steps from $\langle \mathcal{M}_0, \mathcal{M}'_0 \rangle$, $\mathcal{M}$ is last on some execution $\rho \in exec(\mathcal{M}_0)$ of length n, and $\mathcal{S}$ consists exactly of the all models that are last on some execution $\pi \in exec(\mathcal{M}'_0)$ having length *at most* n, and such that ρ is φ-matched by π. Hence, if $\langle \mathcal{M}, \emptyset \rangle$ is reachable, there exists an execution $\rho \in exec(\mathcal{M}_0)$ ending in $\mathcal{M}$, but no execution π of length at most $length(\rho)$ matching ρ. Since in our simulation framework longer executions ρ can only be matched by shorter ones π , there is no execution matching ρ at all, meaning that simulation is violated. On the other hand, if no pair of the form $\langle \mathcal{M}, \emptyset \rangle$ is reachable, then every execution $\rho \in exec(\mathcal{M}_0)$ is φ-matched by some execution π. Note that the completeness of our procedure in exploring all executions $\rho \in exec(\mathcal{M}_0)$ follows from the completeness of the Maude's search command and from our assumption that transition systems are non-blocking. □

Finally, our procedure suggests an approach based on inductive theorem proving to show that a simulation does hold, i.e., that a model transformation preserves operational semantics: inductively prove that terms of the form $\langle \mathcal{M}, \emptyset \rangle$ cannot be reached using Rule (†) from the initial pair $\langle \mathcal{M}_0, \mathcal{M}'_0 \rangle$, using e.g., Maude's prover [7].

5 Conclusion, Related, and Future Work

We present an embedding of essential DSML concepts in Maude. We exploit the rich semantical features of Maude specifications in order to provide models, metamodels, operational semantics, and model transformations with abstract definitions that naturally capture their intuitive meanings. We also give equivalent executable definitions to those concepts, which can be used by Maude for formal verification, and illustrate the approach on a simple example.

Related Works. We build on earlier work [4, 5]. In addition to the representations of models, metamodels, and conformance [4, 5] we define here operational semantics and (operational semantics-preserving) model transformations. The distinction between abstract and equivalent executable definitions is also new.

The closest related works are [1] and [2], who propose different encoding of the syntax and semantics of DSML in Maude. The main difference is that we encode metamodels as MEL specifications, whereas [1] use Maude's sorts, and [2] use an object-oriented extension of Maude. We believe that our approach exploits better the some of the simplest constructions of Maude: MEL order-sorted specifications and their semantics. We also study the semantics preserving model transformaton property, which (to our best knowledge) is new for DSML in Maude. On the other hand, [1] and [2] are more advanced in practical terms; their tools are integrated in the ECLIPSE environment, and they propose higher-level, user-friendly languages for users to define operational semantics, including real- time semantics [10, 11].

Among the many related works, graph transformations are formal modelling languages that have been used for defining semantics of DSML and of model transformations [12, 13, 14], including semantics-preserving model transformations [15, 16]. An advantage of using Maude with respect to these approaches is that they abstract away from attribute values, whereas Maude does not.

A theorem-proving approach for semantical preservation of model transformations is presented in [17]. A theorem-proving approach dedicated to testing model transformations is [18].

Yet another, different approach is taken by the Kermeta frame-

work[4], where methods written in Kermeta's language are *weaved* in a metamodel to make its underlying models executable [19].

The article [20] proposes a rewriting-logic formal semantics for the ATL model transformation language[5].

In the full version of this paper we shall present the application of our approach to an executable version of the SPEM standard[6] and to a transformation to timed Petri nets, borrowed from [17].

In the future we are planing to adapt the general algebraic simulations (and the verification techniques dedicated to them) from the article [6] to semantics-preserving model transformations. Regarding more practical concerns, we are planning to connect our approach to the ECLIPSE environment and to a user-friendly, possibly graphical language for expressing operational semantics.

References

[1] Artur Boronat and José Meseguer. An algebraic semantics for MOF. *Formal Aspects of Computing*, 22(3-4):269–296, 2010.

[2] José Eduardo Rivera, Francisco Durán, and Antonio Vallecillo. Formal specification and analysis of domain specific languages using Maude. *Simulation: Transactions of the Society for Modeling and Simulation International*, 85(11 - 12):778–792, 2009.

[3] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude, A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[4] Marina Egea. *An executable formal semantics for* OCL *with aopplications to model analysis and validation*. PhD thesis, Universidad Complutense de Madrid, 2008.

[5] Maria Egea and Vlad Rusu. Formal executable semantics for conformance in the MDE framework. *Innovations in Systems and Software Engineering*, 6:73– 81, 2010. Extended version of a *UML & FM 2009 Workshop* paper.

[6] José Meseguer, Miguel Palomino, and Narciso Martí-Oliet. Algebraic simulations. *J. Log. Algebr. Program.*, 79(2):103–143, 2010.

[7] Manuel Clavel, Migue Palomino, and Adrián Riesco. Introducing the ITP tool: a tutorial. *Journal of Universal Computer Science*, 12(11):1618–1650, 2006.

[8] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997.

[9] Jan Bergstra and John Tucker. Characterization of computable data types by means of a finite equational specification method. In *International Conference on Automata, Languages and Programming*, volume 81 of *Lecture Notes in Computer Science*, pages 76 – 90, 1980.

[10] José Eduardo Rivera, Cristina Vicente-Chicote, and Antonio Vallecillo. Extending visual modeling languages with timed behavior specifications. In Antonio Brogi, João Araújo, and Raquel Anaya, editors, *CIbSE*, pages 87–100, 2009.

[11] Artur Boronat and Peter Csaba Ölveczky. Formal real-time model transformations in MOMENT2. In David S. Rosenblum and Gabriele Taentzer, editors, *FASE*, volume 6013 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2010.

[12] J. de Lara and H. Vangheluwe. Defining visual notations and their manipulation through meta-modelling and graph transformation. *Journal of Visual Languages and Computing*, 15(3-4):309–330, 2006.

[13] Aditya Agrawal, Gabor Karsai, Sandeep Neema, Feng Shi, and Attila Vizhanyo. The design of a language for model transformations. *Software and System Modeling*, 5(3):261–288, 2006.

[14] György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA - visual automated transformations for formal verification and validation of UML models. In *ASE*, pages 267–270. IEEE Computer Society, 2002.

[15] Guilherme Rangel, Leen Lambers, Barbara König, Hartmut Ehrig, and Paolo Baldan. Behavior preservation in model refactoring using DPO transformations with borrowed contexts. In Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *ICGT*, volume 5214 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2008.

[16] Dénes Bisztray, Reiko Heckel, and Hartmut Ehrig. Verification of architectural refactorings: Rule extraction and tool support. *ECEASST*, 16, 2008.

[17] Benoît Combemale, Xavier Crégut, Pierre-Loïc Garoche, and Xavier Thirioux. Essay on semantics definition in MDE - an instrumented approach for model verification. *Journal of Software*, 4(9):943–958, 2009.

[18] Camillo Fiorentini, Alberto Momigliano, Mario Ornaghi, and Iman Poernomo. A constructive approach to testing model transformations. In *Theory and Practice of Model Transformations, Third International Conference (ICMT'10)*, pages 77–92, 2010.

[19] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented metalanguages. In *MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

[20] Javier Troya and Antonio Vallecillo. Towards a rewriting logic semantics for ATL. In *Theory and Practice of Model Transformations, Third International Conference (ICMT'10)*, pages 230–244, 2010.

[4] `http://www.kermeta.org`
[5] `http://www.eclipse.org/atl/`
[6] `http://www.omg.org/SPEM`