A Metamodel for Maude

José Eduardo Rivera, Francisco Durán, and Antonio Vallecillo

Universidad de Málaga (Spain)
{rivera,duran,av}@lcc.uma.es

Abstract. Models and metamodels play a cornerstone role in Model-Driven Software Development (MDSD). In this paper we introduce a Maude metamodel so that Maude specifications can be represented as models and fully integrated into MDSD processes. Thus, we provide the Maude metamodel specification (implemented as Ecore and KM3 models) and a model-to-text transformation to serialize Maude models (i.e., to get the corresponding Maude code).

1 Metamodel definition

The Maude metamodel is shown at the end of the document. It is based on the Maude language definition [1]. The metamodel does not cover the whole Maude language, it is restricted to the elements needed in our formalization of models and metamodels with Maude. The interested reader is referred to the book [1] for a description of the Maude language concepts.

1.1 Metamodel constraints

Specifying a metamodel implies the choice among different design options. Some of these choices introduce new constraints to the language that must be considered in the metamodel definition. Thus, the following constraints have to be added to those of the Maude language itself [1]:

```
-- Membership axioms cannot have rewrite conditions
context Membership inv:
self.conds->forAll(c | c.oclIsKindOf(EquationalCond))

-- Equations cannot have rewrite conditions
context Equation inv:
self.conds->forAll(c | c.oclIsKindOf(EquationalCond))

-- Functional modules cannot have rules
context FModule inv:
not self.els->exists(e | e.oclIsTypeOf(Rule))

-- Constant terms must be declared (by an operation)
context Constant inv:
Operation.AllInstances()->exists(o | o.name = self.op and self.type.
```

```
  isSubsortOf(o.coarity))

-- Recursive terms must be declared (by an operation)
context RecTerm inv:
Operation.allInstances()->exists(o | o.name = self.op and self.type.
  isSubsortOf(o.coarity) and self.args->forAll(a | a.type.
  isSubsortOf(op.arity->at(self.args->indexOf(a)))))

-- The sort of boolean conditions is sort (or subsort of) 'Bool'
context BooleanCond inv:
let boolSort : Sort = Sort.allInstances->select(s | s.name = 'Bool')
  ->first() in self.lhs.type.isSubsortOf(boolSort)

-- View's operation mappings' attributes cannot be specified.
context View inv:
self.els->select(e | e.oclIsTypeOf(OpTypedMapping))->forAll(m |
  m.atts.oclIsUndefined())

-- Every module expression must refer to a theory or a module
context ModExpression inv:
self.getBaseModExps()->forAll(me | me.oclIsTypeOf(TheoryIdModExp)
  or me.oclIsTypeOf(ModuleIdModExp))

-- Module expressions that refer to modules cannot be composed with those
-- that refer to theories
context CompModExp inv:
self.getBaseModExps()->forAll(me | me.oclIsTypeOf(TheoryIdModExp))
  or self.getBaseModExps()->forAll(me | me.oclIsTypeOf(ModuleIdModExp))

-- The source module expression of a view must refer to a theory
context View inv:
self.from.getBaseModExps()->forAll(me | me.oclIsTypeOf(TheoryIdModExp))

-- A module parameter must refer to a theory
context Parameter inv:
self.modExp.getBaseModExps()->forAll(me | me.oclIsTypeOf(TheoryIdModExp))
```

Operations specification

```
context InstModExp::getBaseModExp() : Bag(ModExpression)
body: self.modExp.getBaseModExp()

context RenModExp::getBaseModExp() : Bag(ModExpression)
body: self.modExp.getBaseModExp()

context ModuleIdModExp::getBaseModExp() : Bag(ModExpression)
body: Bag{self}

context TheoryIdModExp::getBaseModExp() : Bag(ModExpression)
body: Bag{self}
```

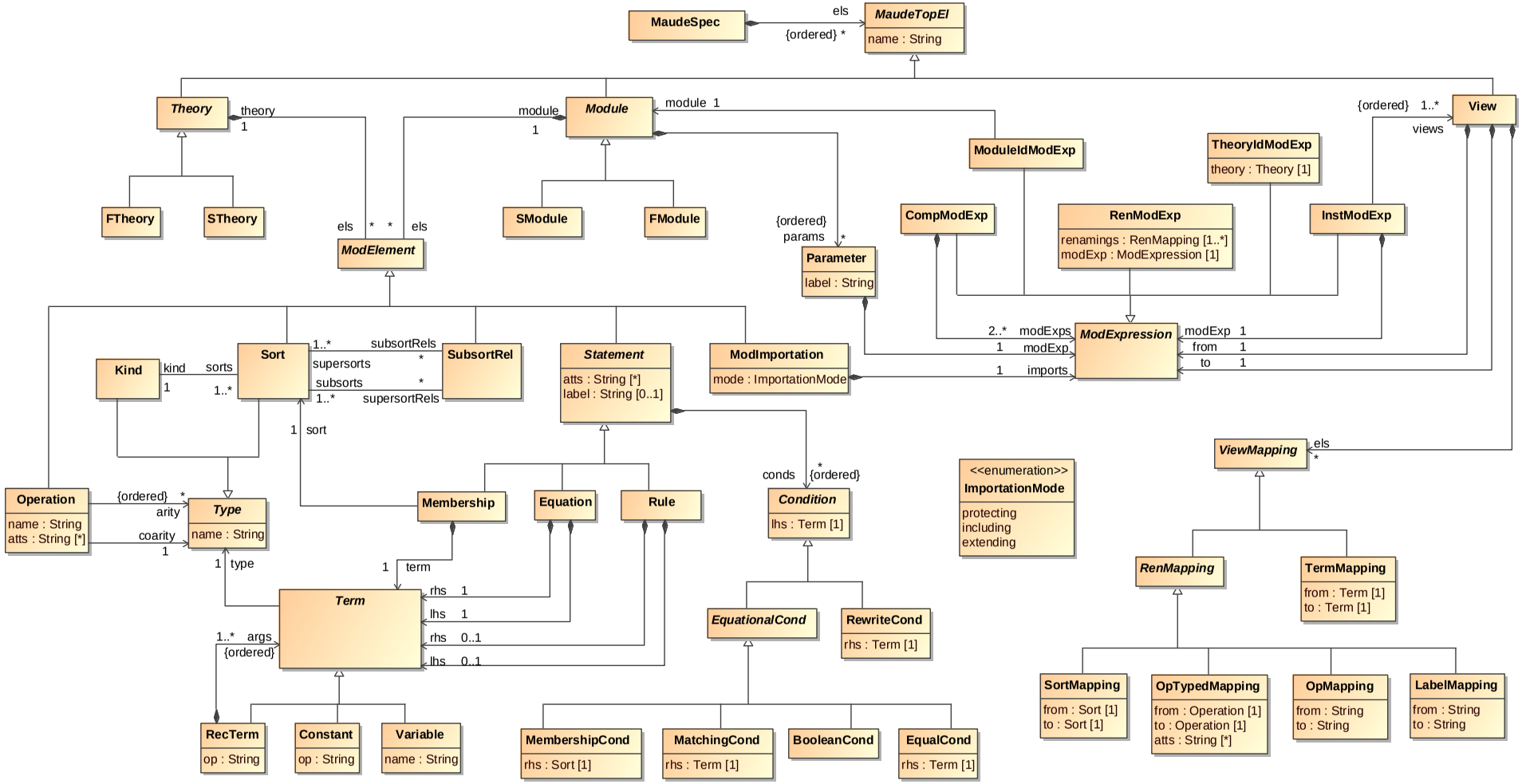
```
context CompModExp::getBaseModExp() : Bag(ModExpression)
body: self.modExps->collect(me | me.getBaseModExp())->flatten()

context Kind::isSubsortOf(st : Type) : Boolean
body: (self = st)

context Sort::isSubsortOf(st : Type) : Boolean
body: (self = st) or self.getAllSupersorts()->exists(s | s = st)

context Sort::getAllSupersorts() : Set(Sort)
body: self.getSupersorts()->union(self.getSupersorts()->collect(s |
  s.getAllSupersorts())->flatten())

context Sort::getSupersorts() : Set(Sort)
body: self.supersortRels->collect(sr | sr.supersorts)->flatten()
```

2 Support

We have developed a set of model-to-text transformations, using Textual Concrete Syntax (TCS, [2]), that allow Maude models to be serialized into the corresponding Maude code. These model-to-text transformations, and the Ecore and KM3 specifications of the Maude metamodel, can be downloaded from http://atenea. lcc.uma.es/index.php/Main_Page/Resources/MaudeMM.

References

1. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude – A High-Performance Logical Framework. Number 4350 in LNCS. Springer, Heidelberg, Germany (2007)
2. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering, New York, NY, USA, ACM (2006) 249–254


MaudeSpec
els
{ordered} *
MaudeTopEl
name : String
Theory
theory
1
module
1
Module
module 1
ModuleIdModExp
TheoryIdModExp
theory : Theory [1]
{ordered} 1..*
views
View
FTheory
STheory
els * * els
ModElement
SModule
FModule
{ordered}
params *
Parameter
label : String
CompModExp
RenModExp
renamings : RenMapping [1..*]
modExp : ModExpression [1]
InstModExp
2..* modExps
1 modExp
ModExpression
modExp 1
from 1
to 1
1 imports
Kind
kind
1
sorts
1..*
Sort
1..* subsortRels
supersorts *
subsorts *
1..* supersortRels
SubsortRel
Statement
atts : String [*]
label : String [0..1]
ModImportation
mode : ImportationMode
1 sort
conds *
{ordered}
Condition
lhs : Term [1]
<<enumeration>>
ImportationMode
protecting
including
extending
ViewMapping
els
*
Operation
name : String
atts : String [*]
{ordered} *
arity
coarity
1
Type
name : String
Membership
Equation
Rule
1 type
1 term
Term
rhs 1
lhs 1
rhs 0..1
lhs 0..1
1..* args
{ordered}
EquationalCond
RewriteCond
rhs : Term [1]
RenMapping
TermMapping
from : Term [1]
to : Term [1]
SortMapping
from : Sort [1]
to : Sort [1]
OpTypedMapping
from : Operation [1]
to : Operation [1]
atts : String [*]
OpMapping
from : String
to : String
LabelMapping
from : String
to : String
RecTerm
op : String
Constant
op : String
Variable
name : String
MembershipCond
rhs : Sort [1]
MatchingCond
rhs : Term [1]
BooleanCond
EqualCond
rhs : Term [1]