Embedding Domain-Specific Modelling Languages into Maude Specifications

Vlad Rusu

INRIA & Laboratoire d'Informatique Fondamentale de Lille, France

Outline

Outline

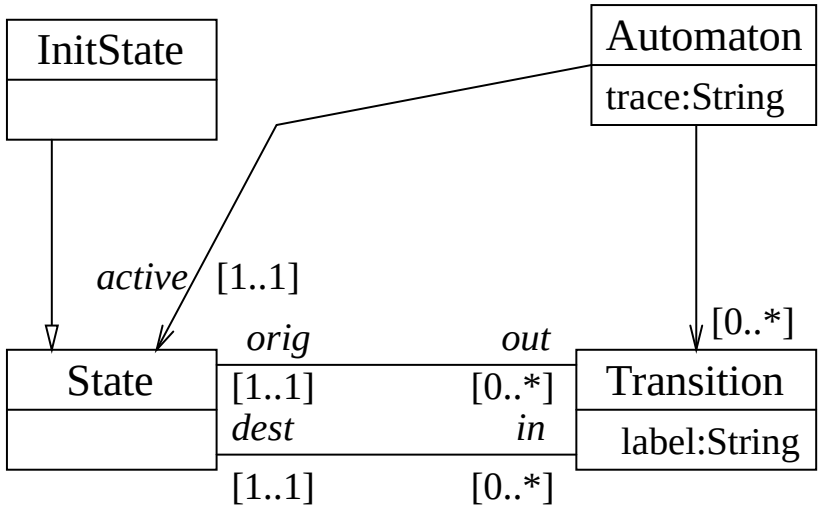
Domain Specific Modelling Languages (DSML)

- What are they:
  - DSML are language for modelling specific domains
  - some DSML are *executable*
- Why are they needed:
  - the Unified Modelling Language (UML) is "too large"
  - people prefer smaller languages adapted to their *domain*
- How to define them:
  - based on *metamodels, models, and model transformations*.

Outline

Syntax of Finite Automata = Metamodel

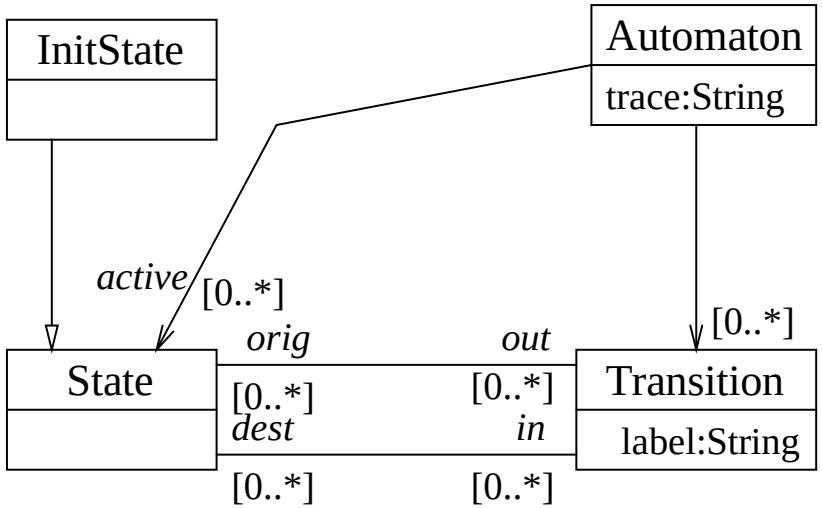
= UML Class Diagram + (optionally) OCL constraints



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

Syntax of Finite Automata = Metamodel

= UML Class Diagram + (optionally) OCL constraints



Context *Transition* : $self.dest.size() = 1$
Context *Automaton* : $self.active.size() = 1$
Context *Transition* : $self.orig.size() = 1$
Context *State* : $self.out \rightarrow forAll(t_1, t_2 | t_1 \neq t_2 \text{ implies } t_1.label \neq t_2.label)$
Context *Transition* : $self.label \neq ""$

Metamodel of Automata as Maude Specification

```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

Metamodel of Automata as Maude Specification

```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

Metamodel of Automata as Maude Specification

```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

Metamodel of Automata as Maude Specification

```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

Metamodel of Automata as Maude Specification

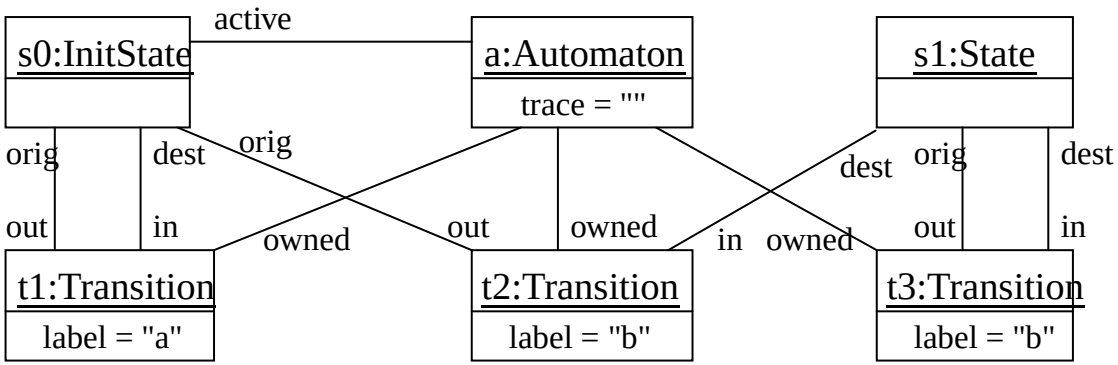
```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

Metamodel of Automata as Maude Specification

```
fth AUTOMATA-META-MODEL is
pr NAT .
pr STRING .

sorts Automaton State InitialState Transition .
subsort InitialState < State .

op trace :  Automaton -> String .  --- attributes
op label :  Transition -> String .

op owned :  Automaton -> Set{Transition}.  --- associations
op orig :  Transition -> Set{State} .
op dest :  Transition -> Set{State} .
op incoming :  State -> Set{Transition} .
op outgoing :  State -> Set{Transition} .
op active :  Automaton -> Set{State} .

eq card(active(X:Automaton)) = 1 -- "implicit" OCL invariant for cardinality ...
eq length(label(W:Transition)) > 0 = true --- "explicit" OCL invariant
...

endfth
```

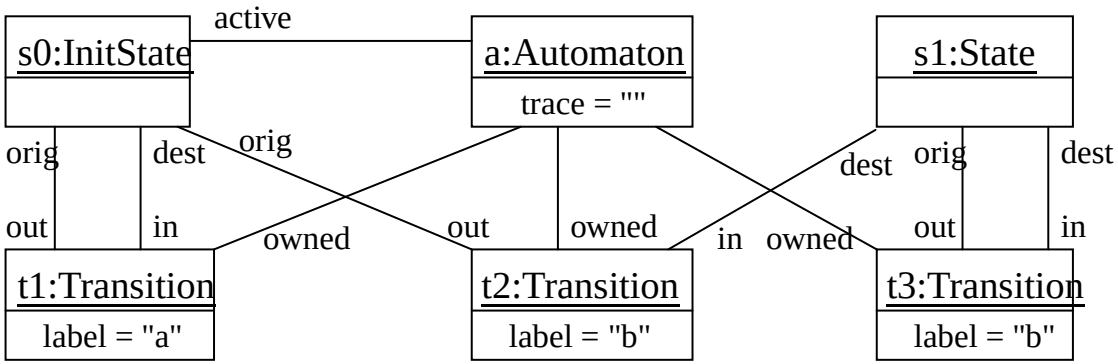
One Automaton = a Model (of the Metamodel)

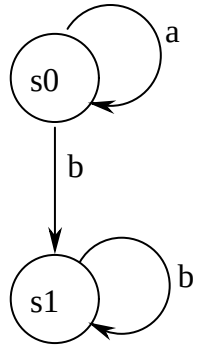
= Object diagram of Class Diagram



One Automaton = a Model (of the Metamodel)

= Object diagram of Class Diagram





Model of an Automaton as Maude Specification

```
fmod AUTOMATON-MODEL is
...includes AUTOMATA-META-MODEL except equations for OCL invariants

op a :  -> Automaton .
op s0 :  -> InitialState .
op s1 :  -> State .
ops t1 t2 t3 :  -> Transition .

eq trace(a) = "" .
eq label(t1) = "a" .
eq label(t2) = "b" .
eq label(t3) = "b" .
eq owned(a) = t1, t2, t3 .
eq active(a) = s0 .
eq orig(t1) = s0 .
eq dest(t1) = s0 .
eq orig(t2) = s0 .
eq dest(t2) = s1 .
eq orig(t3) = s1 .
eq dest(t3) = s1 .
eq incoming(s0) = t1 .
eq outgoing(s0) = t1, t2 .
eq incoming(s1) = t1, t2 .
eq outgoing(s1) = t1 .

endfm
```

Model of an Automaton as Maude Specification

```
fmod AUTOMATON-MODEL is
...includes AUTOMATA-META-MODEL except equations for OCL invariants

op a :  -> Automaton .
op s0 :  -> InitialState .
op s1 :  -> State .
ops t1 t2 t3 :  -> Transition .

eq trace(a) = "" .
eq label(t1) = "a" .
eq label(t2) = "b" .
eq label(t3) = "b" .
eq owned(a) = t1, t2, t3 .
eq active(a) = s0 .
eq orig(t1) = s0 .
eq dest(t1) = s0 .
eq orig(t2) = s0 .
eq dest(t2) = s1 .
eq orig(t3) = s1 .
eq dest(t3) = s1 .
eq incoming(s0) = t1 .
eq outgoing(s0) = t1, t2 .
eq incoming(s1) = t1, t2 .
eq outgoing(s1) = t1 .

endfm
```

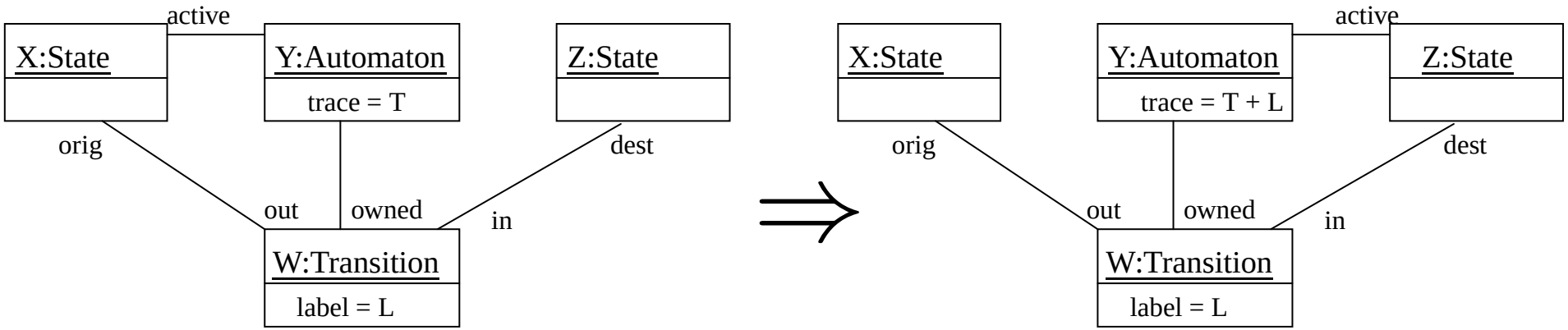
Model of an Automaton as Maude Specification

```
fmod AUTOMATON-MODEL is
...includes AUTOMATA-META-MODEL except equations for OCL invariants

op a :  -> Automaton .
op s0 :  -> InitialState .
op s1 :  -> State .
ops t1 t2 t3 :  -> Transition .

eq trace(a) = "" .
eq label(t1) = "a" .
eq label(t2) = "b" .
eq label(t3) = "b" .
eq owned(a) = t1, t2, t3 .
eq active(a) = s0 .
eq orig(t1) = s0 .
eq dest(t1) = s0 .
eq orig(t2) = s0 .
eq dest(t2) = s1 .
eq orig(t3) = s1 .
eq dest(t3) = s1 .
eq incoming(s0) = t1 .
eq outgoing(s0) = t1, t2 .
eq incoming(s1) = t1, t2 .
eq outgoing(s1) = t1 .

endfm
```

Model of an Automaton as Maude Specification

```
fmod AUTOMATON-MODEL is
...includes AUTOMATA-META-MODEL except equations for OCL invariants

op a :  -> Automaton .
op s0 :  -> InitialState .
op s1 :  -> State .
ops t1 t2 t3 :  -> Transition .

eq trace(a) = "" .
eq label(t1) = "a" .
eq label(t2) = "b" .
eq label(t3) = "b" .
eq owned(a) = t1, t2, t3 .
eq active(a) = s0 .
eq orig(t1) = s0 .
eq dest(t1) = s0 .
eq orig(t2) = s0 .
eq dest(t2) = s1 .
eq orig(t3) = s1 .
eq dest(t3) = s1 .
eq incoming(s0) = t1 .
eq outgoing(s0) = t1, t2 .
eq incoming(s1) = t1, t2 .
eq outgoing(s1) = t1 .

endfm
```

Operational Semantics = Model Transformation
= Graph Rewriting



Automata Semantics as Rewriting over Maude Specs

```
rl
(eq 'active[Y] = X [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = Z [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = T [none] .)
=>
(eq 'active[Y] = Z [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = X [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = '_+_[T, L] [none] .)  .

search [1,10] upModule('AUTOMATON-MODEL,false) =>* M such that
getTrace(M) = '"aaabbb".String
```

Automata Semantics as Rewriting over Maude Specs

```
rl
(eq 'active[Y] = X [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = Z [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = T [none] .)
=>
(eq 'active[Y] = Z [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = X [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = '_+_[T, L] [none] .)  .

search [1,10] upModule('AUTOMATON-MODEL,false) =>* M such that
getTrace(M) = '"aaabbb".String
```

Automata Semantics as Rewriting over Maude Specs

```
rl
(eq 'active[Y] = X [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = Z [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = T [none] .)
=>
(eq 'active[Y] = Z [none] .)
(eq 'owned[Y] = W [none] .)
(eq 'orig[W] = X [none] .)
(eq 'dest[W] = X [none] .)
(eq 'label[W] = L [none] .)
(eq 'trace[Y] = '_+_[T, L] [none] .)  .

search [1,10] upModule('AUTOMATON-MODEL,false) =>* M such that
getTrace(M) = '"aaabbb".String
```

Outline

All DSML can be thus represented!

metamodel $MM \rightsquigarrow$ Maude theory $MM \rightsquigarrow [\![MM]\!]$.
model M of $MM \rightsquigarrow$ Maude module $M_{MM} \rightsquigarrow (\!|M_{MM}|\!)$.

- *conformance*: $(\!|M_{MM}|\!) \in [\![MM]\!]$
- *operational semantics*: rec. function $[\![MM]\!] \rightarrow \mathcal{P}([\![MM]\!])$

are *effectively decidable/computable* in Maude

Also in the paper: *semantics-preserving model transformations*.

All DSML can be thus represented!

metamodel $MM \rightsquigarrow$ Maude theory $MM \rightsquigarrow [\![MM]\!]$.
model M of $MM \rightsquigarrow$ Maude module $M_{MM} \rightsquigarrow (\!|M_{MM}|\!)$.

- *conformance*: $(\!|M_{MM}|\!) \in [\![MM]\!]$
- *operational semantics*: rec. function $[\![MM]\!] \rightarrow \mathcal{P}([\![MM]\!])$

are *effectively decidable/computable* in Maude

Also in the paper: *semantics-preserving model transformations*.

All DSML can be thus represented!

metamodel $MM \rightsquigarrow$ Maude theory $MM \rightsquigarrow [\![MM]\!]$.
model M of $MM \rightsquigarrow$ Maude module $M_{MM} \rightsquigarrow (\!|M_{MM}|\!)$.

- *conformance*: $(\!|M_{MM}|\!) \in [\![MM]\!]$
- *operational semantics*: rec. function $[\![MM]\!] \rightarrow \mathcal{P}([\![MM]\!])$

are *effectively decidable/computable* in Maude

Also in the paper: *semantics-preserving model transformations*.

All DSML can be thus represented!

metamodel $MM \rightsquigarrow$ Maude theory $MM \rightsquigarrow [\![MM]\!]$.
model M of $MM \rightsquigarrow$ Maude module $M_{MM} \rightsquigarrow (\!|M_{MM}|\!)$.

- *conformance*: $(\!|M_{MM}|\!) \in [\![MM]\!]$
- *operational semantics*: rec. function $[\![MM]\!] \rightarrow \mathcal{P}([\![MM]\!])$

are *effectively decidable/computable* in Maude

Also in the paper: *semantics-preserving model transformations*.

All DSML can be thus represented!

metamodel $MM \rightsquigarrow$ Maude theory $MM \rightsquigarrow [\![MM]\!]$.
model M of $MM \rightsquigarrow$ Maude module $M_{MM} \rightsquigarrow (\!|M_{MM}|\!)$.

- *conformance*: $(\!|M_{MM}|\!) \in [\![MM]\!]$
- *operational semantics*: rec. function $[\![MM]\!] \rightarrow \mathcal{P}([\![MM]\!])$

are *effectively decidable/computable* in Maude

Also in the paper: *semantics-preserving model transformations*.

Outline

Related Work

Existing approaches in Maude:

- Mova (Madrid): metamodels as *Maude specifications*
- Moment2 (Leicester) : metamodels as *Maude sorts*
- Maudeling (Málaga) : metamodels as *Full Maude specs*.

Other closely related approaches:

- graph grammars (AGG, Viatra, GreAT. . .)
- the Kermeta framework (class diagrams + methods).

Related Work

Existing approaches in Maude:

- Mova (Madrid): metamodels as *Maude specifications*
- Moment2 (Leicester) : metamodels as *Maude sorts*
- Maudeling (Málaga) : metamodels as *Full Maude specs*.

Other closely related approaches:

- graph grammars (AGG, Viatra, GreAT. . .)
- the Kermeta framework (class diagrams + methods).