Defining Operational Semantics for Domain-Specific Modelling Languages

Vlad Rusu

Inria Lille & Laboratoire d'Informatique Fondamentale de Lille

Outline

Outline

Domain Specific Modelling Languages (DSML)

- Modelling languages are (typically)
  - higher-level than programming languages; graphical
  - based on (parts of) the UML (Unified Modelling Language)
- Why Domain-Specific Modelling Languages?
  - people prefer smaller languages adapted to their *domain*
  - "a DSML is a modelling language designed by its users".

How to help non-specialists define their DSML?

- Defining a language is hard
  - syntax: well-known
  - operational semantics: requires specialised knowledge
- A possible approach to define a language L:
  - choose L' that has a defined operational semantics
  - translate L to L'
  - disadvantages : need to know L'; redone for every L.

How to help non-specialists define their DSML?

- Defining a language is hard
  - syntax: well-known
  - operational semantics: requires specialised knowledge
- A possible approach to define a language L:
  - choose L' that has a defined operational semantics
  - translate L to L'
  - disadvantages : need to know L'; redone for every L.

Alternative: formalise a language design *process*

For DSML, existing process using Model-Driven Engineering (MDE)

- syntax: metamodel; "program"=model of metamodel
- operational semantics: model transformation

Our approach, using Maude:

- formalise models, metamodels, model transformation
- take advantage of Maude's verification tools.

Alternative: formalise a language design *process*

For DSML, existing process using Model-Driven Engineering (MDE)

- syntax: metamodel; "program"=model of metamodel
- operational semantics: model transformation

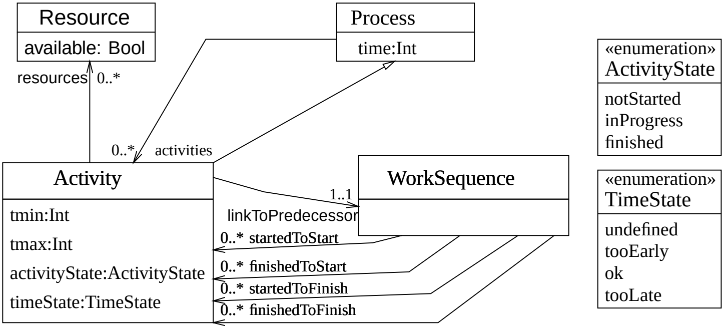
Our approach, using Maude:

- formalise models, metamodels, model transformation
- take advantage of Maude's verification tools.

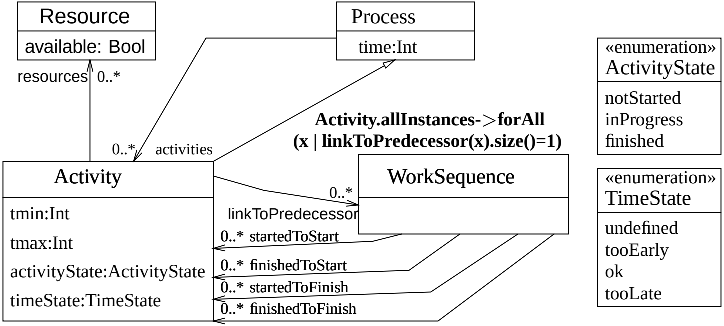
Outline

1. Introduction & Motivation
2. An example of DSML : xSPEM
   - Syntax
   - Semantics
3. The xSPEM example in Maude
   - Syntax
   - Semantics
   - Verification
4. Conclusion, Related, and Future Work

Syntax of xSPEM = Metamodel
= UML Class Diagram + OCL constraints

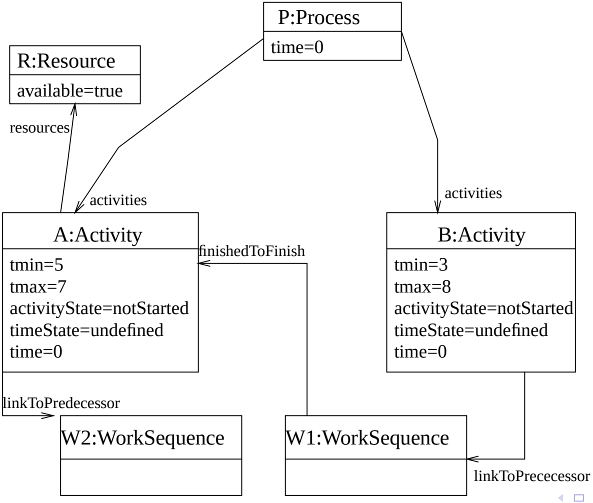


Syntax of xSPEM = Metamodel
= UML Class Diagram + OCL constraints



xSPEM "program" = model of metamodel

= UML object diagram of Class Diagram, satisfying OCL constraints



Time-Passing Rule

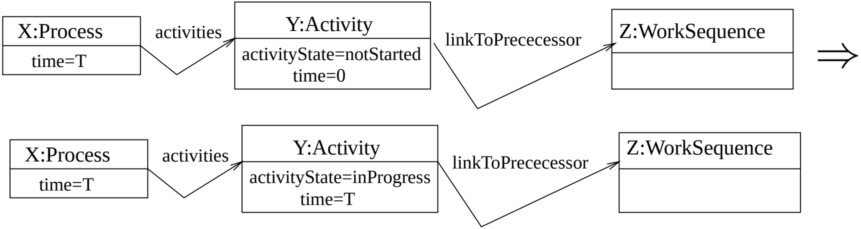
X:Process
time=T

$\Rightarrow$

X:Process
time=T+1

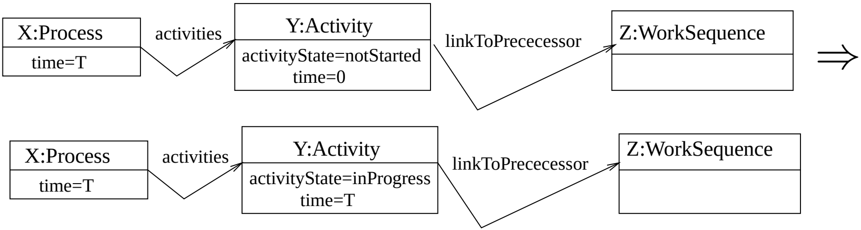
if not (*X oclIsKindOf Activity*)

Starting an Activity



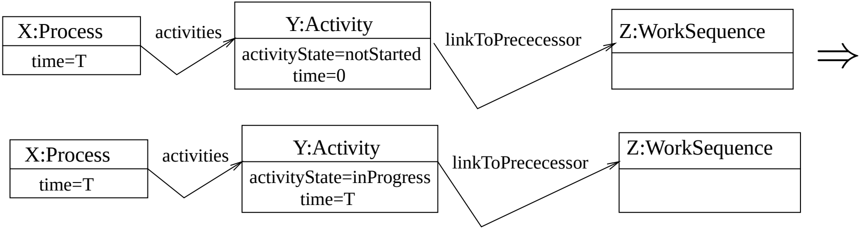
$if\ Z.finishedToStart \rightarrow forAll(u : Activity | u.activityState = finished) \wedge$
$Z.startedToStart \rightarrow forAll(u : Activity | u.activityState = inProgress) \wedge$
$Y.resources \rightarrow forAll(R : Resource | R.available = true) \wedge$
$Y.resources \rightarrow forAll(R : Resource | R.available@Post = false)$

Starting an Activity



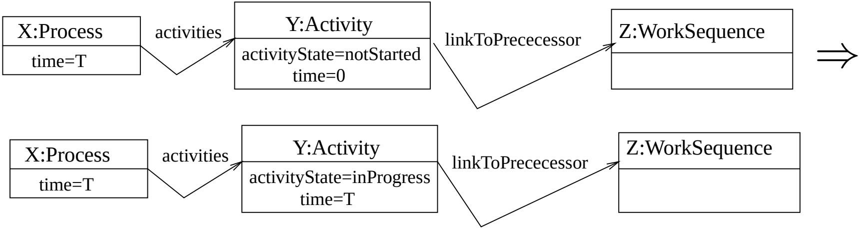
$$\begin{array}{l} \textit{if } Z.\textit{finishedToStart} \rightarrow \textit{forAll}(u : \textit{Activity} \mid u.\textit{activityState} = \textit{finished}) \wedge \\ Z.\textit{startedToStart} \rightarrow \textit{forAll}(u : \textit{Activity} \mid u.\textit{activityState} = \textit{inProgress}) \wedge \\ Y.\textit{resources} \rightarrow \textit{forAll}(R : \textit{Resource} \mid R.\textit{available} = \textit{true}) \wedge \\ Y.\textit{resources} \rightarrow \textit{forAll}(R : \textit{Resource} \mid R.\textit{available@Post} = \textit{false}) \end{array}$$

Starting an Activity



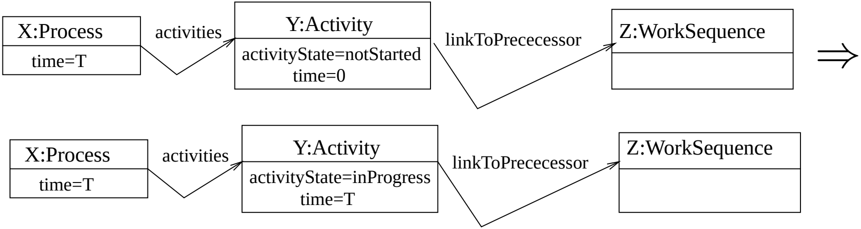
$$\mathit{if}\ Z.\mathit{finishedToStart} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{finished}) \wedge$$
$$Z.\mathit{startedToStart} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{inProgress}) \wedge$$
$$Y.\mathit{resources} \rightarrow \mathit{forAll}(R : \mathit{Resource} | R.\mathit{available} = \mathit{true}) \wedge$$
$$Y.\mathit{resources} \rightarrow \mathit{forAll}(R : \mathit{Resource} | R.\mathit{available@Post} = \mathit{false})$$

Starting an Activity



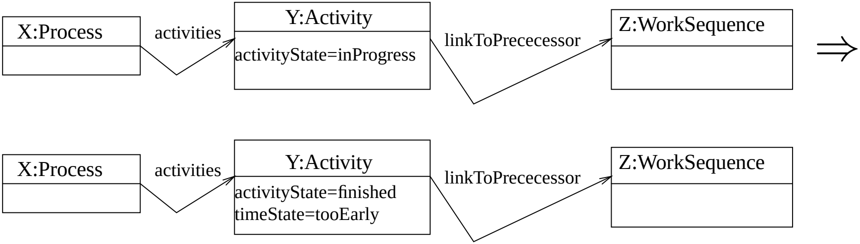
$$\begin{array}{l} \text{if } Z.finishedToStart \rightarrow forAll(u : Activity \mid u.activityState = finished) \wedge \\ Z.startedToStart \rightarrow forAll(u : Activity \mid u.activityState = inProgress) \wedge \\ Y.resources \rightarrow forAll(R : Resource \mid R.available = true) \wedge \\ Y.resources \rightarrow forAll(R : Resource \mid R.available@Post = false) \end{array}$$

Starting an Activity



$$\mathit{if}\ Z.\mathit{finishedToStart} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{finished}) \wedge$$
$$Z.\mathit{startedToStart} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{inProgress}) \wedge$$
$$Y.\mathit{resources} \rightarrow \mathit{forAll}(R : \mathit{Resource} | R.\mathit{available} = \mathit{true}) \wedge$$
$$Y.\mathit{resources} \rightarrow \mathit{forAll}(R : \mathit{Resource} | R.\mathit{available@Post} = \mathit{false})$$

Finishing an Activity: too early

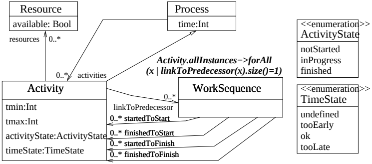


$$
\begin{aligned}
&\mathit{if}\ Z.\mathit{startedToFinish} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{inProgress}) \wedge \\
&Z.\mathit{finishedToFinish} \rightarrow \mathit{forAll}(u : \mathit{Activity} | u.\mathit{activityState} = \mathit{finished}) \wedge \\
&X.\mathit{time} - Y.\mathit{time} < Y.\mathit{tmin} \wedge \\
&Y.\mathit{Resources} \rightarrow \mathit{forAll}(R : \mathit{Resource} | R.\mathit{available@Post} = \mathit{true})
\end{aligned}
$$

Outline

Metamodel

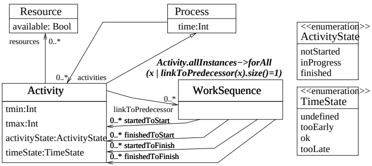
```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm
```



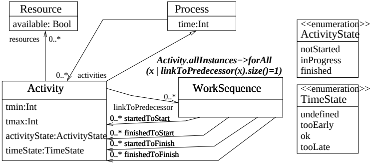

```
spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                       WorkSequence -> Set{Activity} .
endspec
```

Metamodel

```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                       WorkSequence -> Set{Activity} .
endspec
```

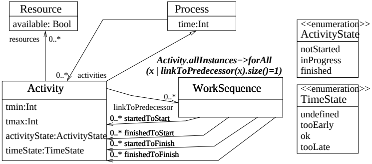


Metamodel




```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                       WorkSequence -> Set{Activity} .
endspec
```

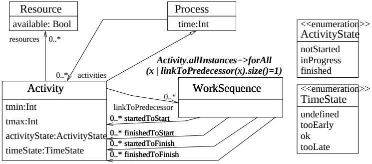
Metamodel



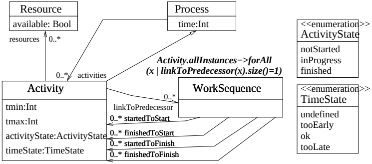

```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished:->ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                      WorkSequence -> Set{Activity} .
endspec
```

Metamodel

```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                       WorkSequence -> Set{Activity} .
endspec
```

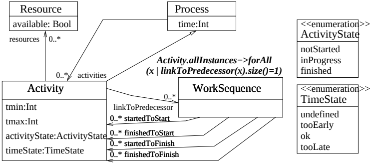


Metamodel




```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                 WorkSequence -> Set{Activity} .
endspec
```

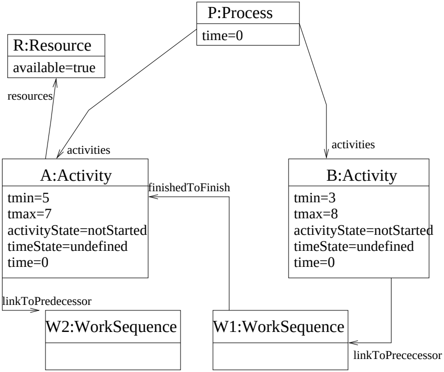
Metamodel




```
fmod ACTIVITYSTATE is
sort ActivityState .
ops notStarted inProgress finished : -> ActivityState .
endfm
fmod TIMESTATE is
sort TimeState .
ops tooEarly ok tooLate undefined :  -> TimeState .
endfm

spec xSPEM-METAMODEL is
protecting ACTIVITYSTATE + TIMESTATE + BOOL + INT .
sorts Process Activity WorkSequence Resource .
subsort Activity < Process .
-- ...  sets of Process, Activity, WorkSequence, Resource...

op time :  Process -> Int .
op activitites :  Process -> Set{Activity} .
ops tmin tmax :  Activity -> Int .
op activityState :  Activity -> ActivityState .
op timeState :  Activity -> TimeState .
op resources :  Activity -> Set{Resource} .
op linkToPredecessor :  Activity -> Set{WorkSequence} .
op available :  resource -> Bool .
-- OCL constraint for cardinality of linkToPredecessor
eq card(linkToPredecessor(x:Activity)) = 1 .
ops startedToStart finishedToFinish startedToFinish finishedToStart :
                                                      WorkSequence -> Set{Activity} .
endspec
```

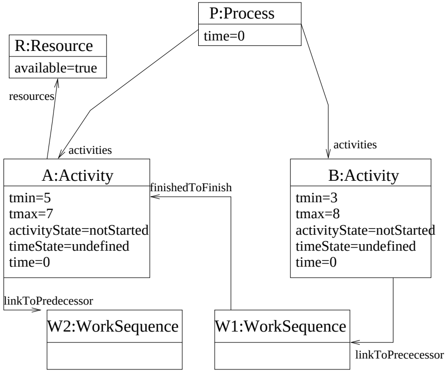
Model

```
fmod xSPEM-MODEL is
extending xSPEM-METAMODEL .
op P : -> Process .
ops A B : -> Activity .
ops W1 W2 :  -> WorkSequence .
op R : -> Resource
eq time(P) = 0 .
eq activities(P) = A, B .
eq tmin(A) = 5 .
eq tmax(A) = 7 .
eq time(A) = 0 .
eq activityState(A) = notStarted .
eq timeState(A) = undefined .
eq activities(A) = empty .
eq linkToPredecessor(A) = W2 .
eq resources(A) = R .
eq available(R) = true .
eq tmin(B) = 3 .
eq tmax(B) = 8 .
eq time(B) = 0 .
eq activityState(B) = notStarted .
eq timeState(B) = undefined .
eq linkToPredecessor(B) = W1 .
eq resources(B) = empty.
eq activities(B) = empty .
eq finishedToFinish(W1) = A .
eq startedToFinish(W1) = empty .
eq finishedToStart(W1) = empty .
eq startedToStart(W1) = empty .
eq finishedToFinish(W2) = empty.
eq startedToFinish(W2) = empty .
eq finishedToStart(W2) = empty .
eq startedToStart(W2) = empty .
endfm
```



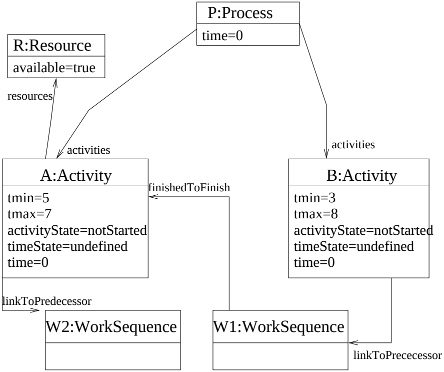
Model

```
fmod xSPEM-MODEL is
extending xSPEM-METAMODEL .
op P : -> Process .
ops A B : -> Activity .
ops W1 W2 :  -> WorkSequence .
op R : -> Resource
eq time(P) = 0 .
eq activities(P) = A, B .
eq tmin(A) = 5 .
eq tmax(A) = 7 .
eq time(A) = 0 .
eq activityState(A) = notStarted .
eq timeState(A) = undefined .
eq activities(A) = empty .
eq linkToPredecessor(A) = W2 .
eq resources(A) = R .
eq available(R) = true .
eq tmin(B) = 3 .
eq tmax(B) = 8 .
eq time(B) = 0 .
eq activityState(B) = notStarted .
eq timeState(B) = undefined .
eq linkToPredecessor(B) = W1 .
eq resources(B) = empty.
eq activities(B) = empty .
eq finishedToFinish(W1) = A .
eq startedToFinish(W1) = empty .
eq finishedToStart(W1) = empty .
eq startedToStart(W1) = empty .
eq finishedToFinish(W2) = empty.
eq startedToFinish(W2) = empty .
eq finishedToStart(W2) = empty .
eq startedToStart(W2) = empty .
endfm
```



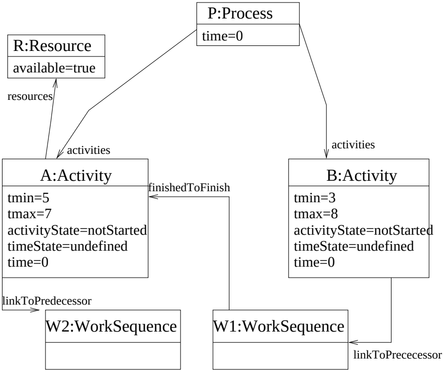
Model

```
fmod xSPEM-MODEL is
extending xSPEM-METAMODEL .
op P : -> Process .
ops A B : -> Activity .
ops W1 W2 :  -> WorkSequence .
op R : -> Resource
eq time(P) = 0 .
eq activities(P) = A, B .
eq tmin(A) = 5 .
eq tmax(A) = 7 .
eq time(A) = 0 .
eq activityState(A) = notStarted .
eq timeState(A) = undefined .
eq activities(A) = empty .
eq linkToPredecessor(A) = W2 .
eq resources(A) = R .
eq available(R) = true .
eq tmin(B) = 3 .
eq tmax(B) = 8 .
eq time(B) = 0 .
eq activityState(B) = notStarted .
eq timeState(B) = undefined .
eq linkToPredecessor(B) = W1 .
eq resources(B) = empty.
eq activities(B) = empty .
eq finishedToFinish(W1) = A .
eq startedToFinish(W1) = empty .
eq finishedToStart(W1) = empty .
eq startedToStart(W1) = empty .
eq finishedToFinish(W2) = empty.
eq startedToFinish(W2) = empty .
eq finishedToStart(W2) = empty .
eq startedToStart(W2) = empty .
endfm
```



Model

```
fmod xSPEM-MODEL is
extending xSPEM-METAMODEL .
op P : -> Process .
ops A B : -> Activity .
ops W1 W2 :  -> WorkSequence .
op R : -> Resource
eq time(P) = 0 .
eq activities(P) = A, B .
eq tmin(A) = 5 .
eq tmax(A) = 7 .
eq time(A) = 0 .
eq activityState(A) = notStarted .
eq timeState(A) = undefined .
eq activities(A) = empty .
eq linkToPredecessor(A) = W2 .
eq resources(A) = R .
eq available(R) = true .
eq tmin(B) = 3 .
eq tmax(B) = 8 .
eq time(B) = 0 .
eq activityState(B) = notStarted .
eq timeState(B) = undefined .
eq linkToPredecessor(B) = W1 .
eq resources(B) = empty.
eq activities(B) = empty .
eq finishedToFinish(W1) = A .
eq startedToFinish(W1) = empty .
eq finishedToStart(W1) = empty .
eq startedToStart(W1) = empty .
eq finishedToFinish(W2) = empty.
eq startedToFinish(W2) = empty .
eq finishedToStart(W2) = empty .
eq startedToStart(W2) = empty .
endfm
```



Conformance

What does "model-to-metamodel conformance" mean?

Ideally, that model "belongs to" metamodel. Let $[\![\mathcal{MM}]\!]$ be the *set of algebras* of (the Maude specificiation of) $\mathcal{MM}$ such that

- A interprets protected modules as their initial algebra
- A interprets sorts denoting classes c by finite sets $A(c)$
- $A(c_1) \cap A(c_2) = \emptyset$ if c_1, c_2 do not inherit from each other.

Let $(\!|\mathcal{M}_{\mathcal{MM}}|\!)$ be the *initial algebra* of (the Maude module for) $\mathcal{M}$.

Conformance (abstract) $\mathcal{M} : \mathcal{MM}$ if $(\!|\mathcal{M}_{\mathcal{MM}}|\!) \in [\![\mathcal{MM}]\!]$.

Conformance

What does "model-to-metamodel conformance" mean?

Ideally, that model "belongs to" metamodel. Let $[\![\mathcal{MM}]\!]$ be the *set of algebras* of (the Maude specificiation of) $\mathcal{MM}$ such that

- A interprets protected modules as their initial algebra
- A interprets sorts denoting classes c by finite sets $A(c)$
- $A(c_1) \cap A(c_2) = \emptyset$ if c_1, c_2 do not inherit from each other.

Let $(\!|\mathcal{M}_{\mathcal{MM}}|\!)$ be the *initial algebra* of (the Maude module for) $\mathcal{M}$.

Conformance (abstract) $\mathcal{M} : \mathcal{MM}$ if $(\!|\mathcal{M}_{\mathcal{MM}}|\!) \in [\![\mathcal{MM}]\!]$.

Conformance

What does "model-to-metamodel conformance" mean?

Ideally, that model "belongs to" metamodel. Let $[\![\mathcal{MM}]\!]$ be the *set of algebras* of (the Maude specificiation of) $\mathcal{MM}$ such that

- A interprets protected modules as their initial algebra
- A interprets sorts denoting classes c by finite sets $A(c)$
- $A(c_1) \cap A(c_2) = \emptyset$ if c_1, c_2 do not inherit from each other.

Let $(\!|\mathcal{M}_{\mathcal{MM}}|\!)$ be the *initial algebra* of (the Maude module for) $\mathcal{M}$.

Conformance (abstract) $\mathcal{M} : \mathcal{MM}$ if $(\!|\mathcal{M}_{\mathcal{MM}}|\!) \in [\![\mathcal{MM}]\!]$.

Conformance

What does "model-to-metamodel conformance" mean?

Ideally, that model "belongs to" metamodel. Let $[\![\mathcal{MM}]\!]$ be the *set of algebras* of (the Maude specificiation of) $\mathcal{MM}$ such that

- A interprets protected modules as their initial algebra
- A interprets sorts denoting classes c by finite sets $A(c)$
- $A(c_1) \cap A(c_2) = \emptyset$ if c_1, c_2 do not inherit from each other.

Let $(\!|\mathcal{M}_{\mathcal{MM}}|\!)$ be the *initial algebra* of (the Maude module for) $\mathcal{M}$.

Conformance (abstract) $\mathcal{M} : \mathcal{MM}$ if $(\!|\mathcal{M}_{\mathcal{MM}}|\!) \in [\![\mathcal{MM}]\!]$.

Conformance

How to actually check it?

Executable version of conformance: evaluate equations (denoting OCL constraints) of $\mathcal{MM}$ in $\mathcal{M}$, check that all hold

(Church-Rosser equations - proved in PhD of Marina Egea).

Conformance (abstract) $\equiv$ Conformance (executable)

Moreover, (†) $[\![\mathcal{MM}]\!] \simeq \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$.

Conformance

How to actually check it?

Executable version of conformance: evaluate equations (denoting OCL constraints) of $\mathcal{MM}$ in $\mathcal{M}$, check that all hold

(Church-Rosser equations - proved in PhD of Marina Egea).

Conformance (abstract) $\equiv$ Conformance (executable)

Moreover, $(\dagger)$ $[\![\mathcal{MM}]\!] \simeq \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$.

Conformance

How to actually check it?

Executable version of conformance: evaluate equations (denoting OCL constraints) of $\mathcal{MM}$ in $\mathcal{M}$, check that all hold

(Church-Rosser equations - proved in PhD of Marina Egea).

Conformance (abstract) $\equiv$ **Conformance (executable)**

Moreover, ($\dagger$) $[\![\mathcal{MM}]\!] \simeq \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$.

Operational Semantics: Abstract & Executable

Op. Sem. (abstract) $\{F : [\![\mathcal{MM}]\!] \rightarrow \mathcal{P}_f([\![\mathcal{MM}]\!]) | F \ recursive\}$

by (†), equivalent to

$\{F : \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\} \rightarrow \mathcal{P}_f(\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}) | F \ recursive\}$

Use Maude's reflection of $\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$ as sort `MM` + Bergstra & Tucker + def. of *Set* $\rightsquigarrow$ first *executable* definition

$\{\mathtt{F} : \mathtt{MM} \rightarrow \mathtt{Set\{MM\}} | \mathtt{F} \ \textit{defined by Church-Rosser equations}\}$

Using $\mathtt{y} \in \mathtt{F(x)}$ iff $[\mathtt{x}] \Rightarrow [\mathtt{y}]$ `if` $\mathtt{y}, \mathtt{z} := \mathtt{F(x)}$

Op. Sem. (executable) = any rewrite relation of a set of (executable) rewrite rules over the sort `MM` reflecting $[\![\mathcal{MM}]\!]$.

Operational Semantics: Abstract & Executable

Op. Sem. (abstract) $\{F : [\![\mathcal{MM}]\!] \to \mathcal{P}_f([\![\mathcal{MM}]\!]) | F \textit{ recursive}\}$

by (†), equivalent to

$$\{F : \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\} \to \mathcal{P}_f(\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}) | F \textit{ recursive}\}$$

Use Maude's reflection of $\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$ as sort `MM` + Bergstra & Tucker + def. of *Set* $\rightsquigarrow$ first *executable* definition

$$\{\mathtt{F} : \mathtt{MM} \to \mathtt{Set\{MM\}} | \mathtt{F} \textit{ defined by Church-Rosser equations}\}$$

$$\text{Using } \mathtt{y} \in \mathtt{F(x)} \text{ iff } [\mathtt{x}] \Rightarrow [\mathtt{y}] \ \mathtt{if}\ \mathtt{y,z := F(x)}$$

Op. Sem. (executable) = any rewrite relation of a set of (executable) rewrite rules over the sort `MM` reflecting $[\![\mathcal{MM}]\!]$.

Operational Semantics: Abstract & Executable

Op. Sem. (abstract) $\{F : [\![\mathcal{MM}]\!] \to \mathcal{P}_f([\![\mathcal{MM}]\!]) | F \textit{ recursive}\}$

by (†), equivalent to

$\{F : \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\} \to \mathcal{P}_f(\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}) | F \textit{ recursive}\}$

Use Maude's reflection of $\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$ as sort `MM` + Bergstra & Tucker + def. of *Set* $\rightsquigarrow$ first *executable* definition

$\{\mathtt{F} : \mathtt{MM} \to \mathtt{Set\{MM\}} | \mathtt{F} \textit{ defined by Church-Rosser equations}\}$

Using $\mathtt{y} \in \mathtt{F(x)}$ iff $[\mathtt{x}] \Rightarrow [\mathtt{y}]$ `if` $\mathtt{y,z := F(x)}$

Op. Sem. (executable) = any rewrite relation of a set of (executable) rewrite rules over the sort `MM` reflecting $[\![\mathcal{MM}]\!]$.

Operational Semantics: Abstract & Executable

Op. Sem. (abstract) $\{F : [\![\mathcal{MM}]\!] \to \mathcal{P}_f([\![\mathcal{MM}]\!]) | F \textit{ recursive}\}$

by (†), equivalent to

$$\{F : \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\} \to \mathcal{P}_f(\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}) | F \textit{ recursive}\}$$

Use Maude's reflection of $\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$ as sort `MM` + Bergstra & Tucker + def. of *Set* $\rightsquigarrow$ first *executable* definition

$$\{\mathtt{F} : \mathtt{MM} \to \mathtt{Set}\{\mathtt{MM}\} | \mathtt{F} \textit{ defined by Church-Rosser equations}\}$$

Using $\mathtt{y} \in \mathtt{F(x)}$ iff $[\mathtt{x}] \Rightarrow [\mathtt{y}]$ `if` $\mathtt{y}, \mathtt{z} := \mathtt{F(x)}$

Op. Sem. (executable) = any rewrite relation of a set of (executable) rewrite rules over the sort `MM` reflecting $[\![\mathcal{MM}]\!]$.

Operational Semantics: Abstract & Executable

Op. Sem. (abstract) $\{F : [\![\mathcal{MM}]\!] \to \mathcal{P}_f([\![\mathcal{MM}]\!]) | F \textit{ recursive}\}$

by (†), equivalent to

$$\{F : \{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\} \to \mathcal{P}_f(\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}) | F \textit{ recursive}\}$$

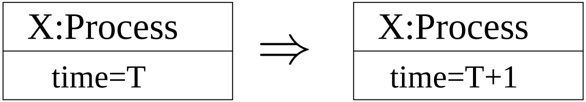
Use Maude's reflection of $\{\mathcal{M}_{\mathcal{MM}} | \mathcal{M} : \mathcal{MM}\}$ as sort `MM` + Bergstra & Tucker + def. of *Set* $\rightsquigarrow$ first *executable* definition

$$\{\mathtt{F} : \mathtt{MM} \to \mathtt{Set\{MM\}} | \mathtt{F} \textit{ defined by Church-Rosser equations}\}$$

$$\text{Using } \mathtt{y} \in \mathtt{F(x)} \text{ iff } [\mathtt{x}] \Rightarrow [\mathtt{y}] \ \mathtt{if}\ \mathtt{y}, \mathtt{z} := \mathtt{F(x)}$$

Op. Sem. (executable) = any rewrite relation of a set of (executable) rewrite rules over the sort `MM` reflecting $[\![\mathcal{MM}]\!]$.

Time-passing rule

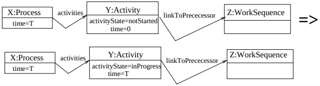


if not $($*X oclIsKindOf Activity*$)$

```
crl
(eq 'time[X:Term]=T:Term .)
  =>
(eq 'time[X:Term]= '_+_[T:Term,'s_['0.Zero]] .)
 if not downTerm(X:Term, errorProcess) ::  Activity .
```

Starting an Activity
(finishing activities: similar)

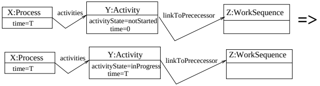
```
crl
M => M'
if
((eq 'time[X:Term] = T:Term .)
(eq 'activities[X:Term] = L:Term .)
(eq 'activityState[Y:Term] = 'notStarted.ActivityState .)
(eq 'time[Y:Term] = '0.Zero .)
(eq 'linkToPredecessor[Y:Term] = W:Term .)
ES:EquationSet) := getEqs(M) ∧
downTerm(Y:Term,ErrorAct) in downTerm(L:Term,ErrorAct) ∧
 forAll1(M,startedToStart(downTerm(W:Term,ErrorWorkSeq))) ∧
forAll2(M,finishedToStart(downTerm(W:Term,ErrorWorkSeq))) ∧
forAll3(M,resources(downTerm(Y:Term, ErrorAct))) ∧
M':Module := forAll4(setEquations(M,
((eq 'time[X:Term] = T:Term .)
(eq 'activities[X:Term] = L:Term .)
(eq 'activityState[Y:Term] = 'inProgress.ActivityState .)
(eq 'time[Y:Term] = T:Term .)
(eq 'linkToPredecessor[Y:Term] = W:Term .)
ES:EquationSet),resources(downTerm(Y:Term, ErrorAct)))) .
```



if Z.finishedToStart –> forAll(u:Activity| u.activityState = finished) /\
Z.startedToStart –> forAll(u:Activity| u.activityState = inProgress)} /\
Y.resources –> forAll(R:Resource | R.available = true)/\
Y.resources –> forAll(R:Resource | R.available@Post = false)

Starting an Activity
(finishing activities: similar)

```
crl
M => M'
if
(('eq 'time[X:Term] = T:Term .)
(eq 'activities[X:Term] = L:Term .)
(eq 'activityState[Y:Term] = 'notStarted.ActivityState .)
(eq 'time[Y:Term] = '0.Zero .)
(eq 'linkToPredecessor[Y:Term] = W:Term .)
ES:EquationSet) := getEqs(M) ∧
downTerm(Y:Term,ErrorAct) in downTerm(L:Term,ErrorAct) ∧
 forAll1(M,startedToStart(downTerm(W:Term,ErrorWorkSeq))) ∧
forAll2(M,finishedToStart(downTerm(W:Term,ErrorWorkSeq))) ∧
forAll3(M,resources(downTerm(Y:Term, ErrorAct))) ∧
M':Module := forAll4(setEquations(M,
(('eq 'time[X:Term] = T:Term .)
(eq 'activities[X:Term] = L:Term .)
(eq 'activityState[Y:Term] = 'inProgress.ActivityState .)
(eq 'time[Y:Term] = T:Term .)
(eq 'linkToPredecessor[Y:Term] = W:Term .)
ES:EquationSet),resources(downTerm(Y:Term, ErrorAct)))) .
```



if Z.finishedToStart –> forAll(u:Activity| u.activityState = finished) /\
Z.startedToStart –> forAll(u:Activity| u.activityState = inProgress)} /\
Y.resources –> forAll(R:Resource | R.available = true)/\
Y.resources –> forAll(R:Resource | R.available@Post = false)

Verification

Are models with all activities finished in due time reachable ?

```
search[1] upModule('xSPEM-MODEL,false)=>*M
such that allFinished(M) and allOK(M).
```

- tmin(A)=5, tmax(A)=7, tmin(B)=3, tmax(B)=8: immediate
- 41 secs if multiplied by 10
- 10 mins 58 secs if multiplied by 20 . . .

Verification

Are models with all activities finished in due time reachable ?

```
search[1] upModule('xSPEM-MODEL,false)=>*M
such that allFinished(M) and allOK(M).
```

- tmin(A)=5, tmax(A)=7, tmin(B)=3, tmax(B)=8: immediate
- 41 secs if multiplied by 10
- 10 mins 58 secs if multiplied by 20 . . .

Verification

Are models with all activities finished in due time reachable ?

```
search[1] upModule('xSPEM-MODEL,false)=>*M
such that allFinished(M) and allOK(M).
```

- tmin(A)=5, tmax(A)=7, tmin(B)=3, tmax(B)=8: immediate
- 41 secs if multiplied by 10
- 10 mins 58 secs if multiplied by 20 . . .

Outline

Conclusion, Related, and Future Work

- Formalisation of MDE approach to defining DSML
- For more information: http://researchers.lille.inria.fr/~rusu/SoSym/paper.pdf
- Existing approaches in Maude:
  - Moment2 (Leicester) : metamodels as (base level) sorts
  - Maudeling (Málaga) : metamodels as O-O modules
- Other approaches: graph grammars; Kermeta (Inria); . . .
- Current : implementation in K/Maude, a framework for operational semantics definition.