LANGUAGE ENGINEERING WITH THE GEMOC STUDIO

Tutorial @ MODELS 2017, September 18th, 2017

Benoit Combemale (Univ. Toulouse)
http://www.combemale.fr
benoit.combemale@irit.fr
@bcombemale

Julien Deantoni (Univ. Nice-Sophia-Antipolis)
http://www.i3s.unice.fr/~deantoni/
julien.deantoni@polytech.unice.fr

Add concurrency in your semantics !

https://github.com/gemoc/MODELS2017Tutorial

Reifying Concurrency in xDSML: Limitations

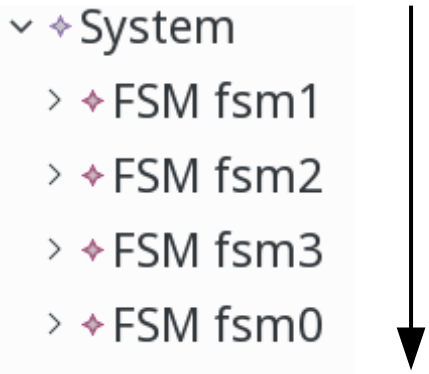
- Concurrency remains implicit and ad-hoc in language design and implementation:
  - Design: implicitly inherited from the meta-language used
  - Implementation: mostly embedded in the underlying execution environment

- The lack of an explicit concurrency specification in language design prevents:
  - leveraging the concurrency concern of a particular domain or platform
  - a complete understanding of the behavioral semantics
  - effective concurrency-aware analysis techniques
  - effective techniques for producing semantic variants
  - analysis of the deployment on parallel architectures

Reifying Concurrency in xDSML: Limitations

```
@Main
def public void main() {
for(FSM sm : _self.ownedFsms){
    if (! sm.inputBuffer.isEmpty){
        sm.run()
        anFSMRan = true
    }
}
```

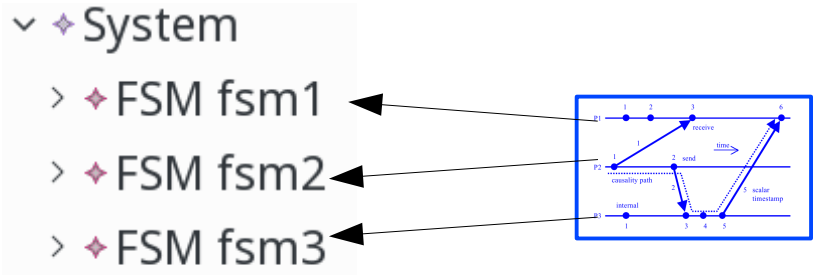
Scheduling order depends on the "ownedFsms" collection



Iterate over the collection

☹ Copy and paste changes the order and eventually the execution

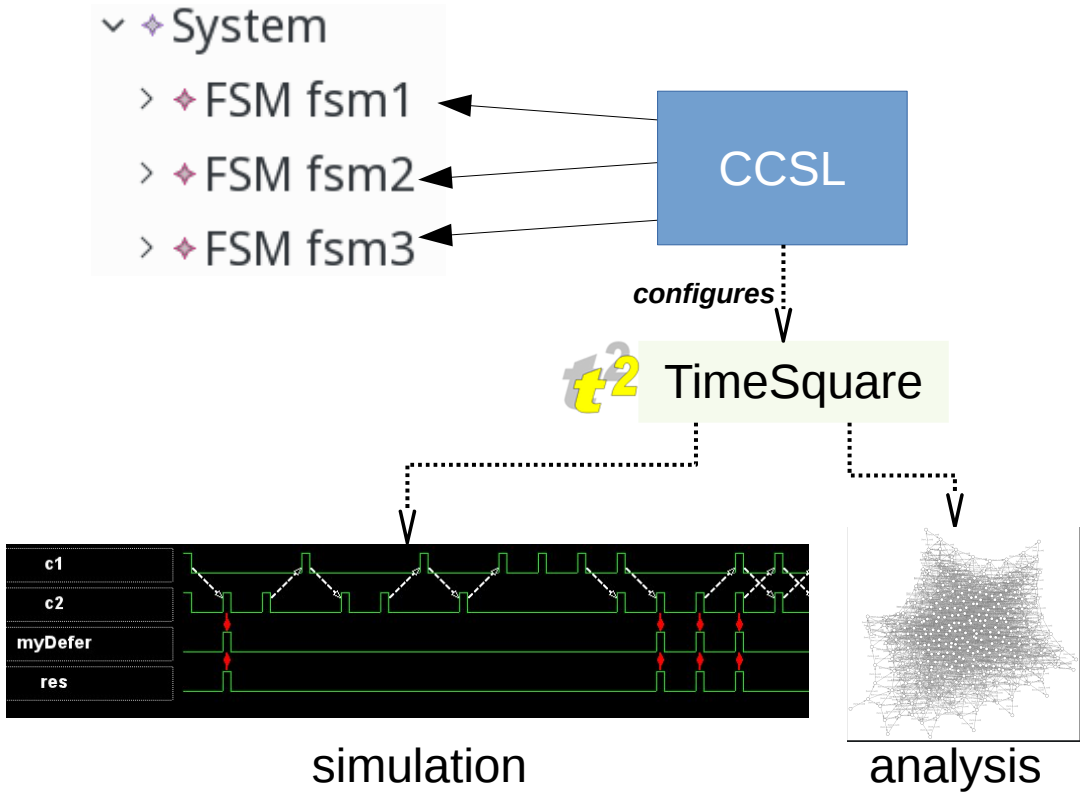
Reifying Concurrency in xDSML: Proposition




```
@Main
def public void main() {
 for(FSM sm : _self.ownedFsms){
     if (! sm.inputBuffer.isEmpty){
         sm.run()
         anFSMRan = true
     }
 }
```

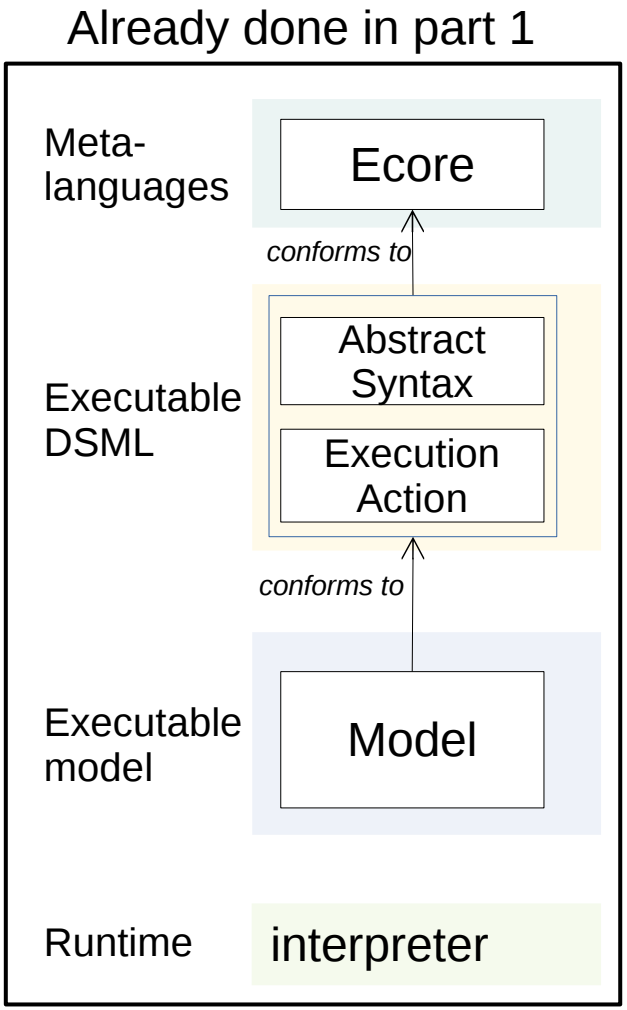
We can use ***Logical Time*** to specify the **partial ordering** that are correct according to our semantics, in a **platform independent** point of view

Reifying Concurrency in xDSML: Proposition

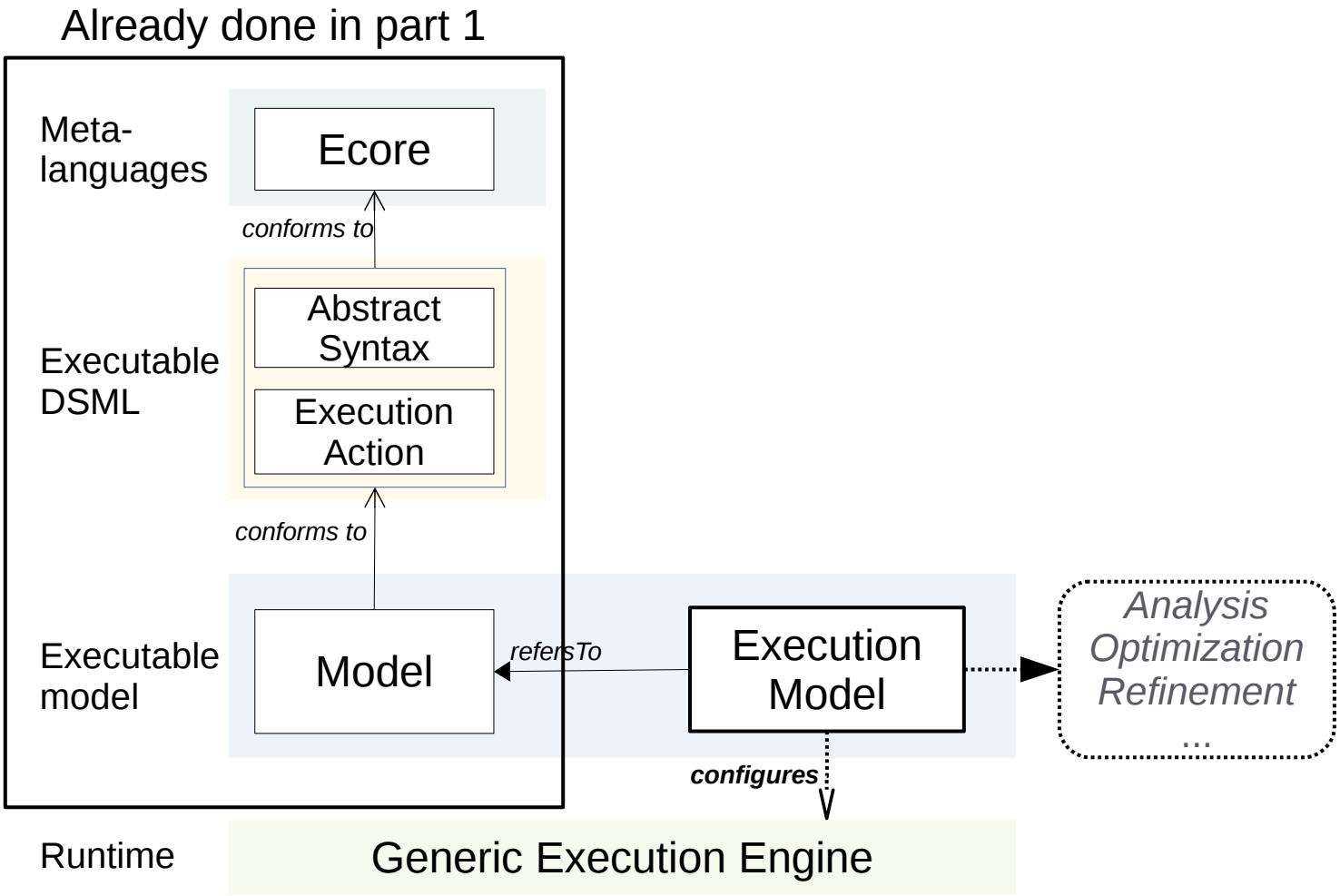


We can use ***Logical Time*** to specify the **partial ordering** that are correct according to our semantics, in a **platform independent** point of view

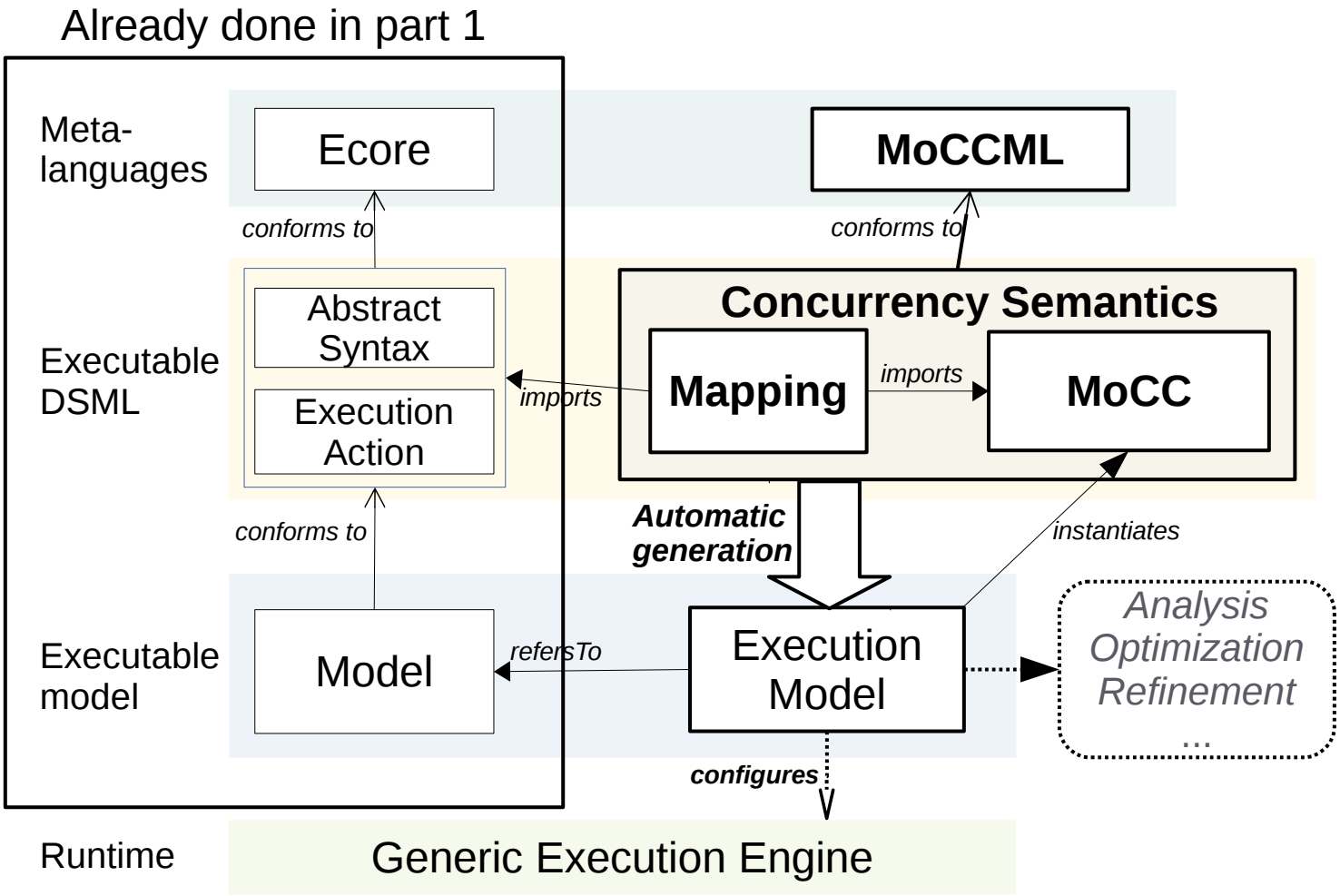
The MoCCML approach



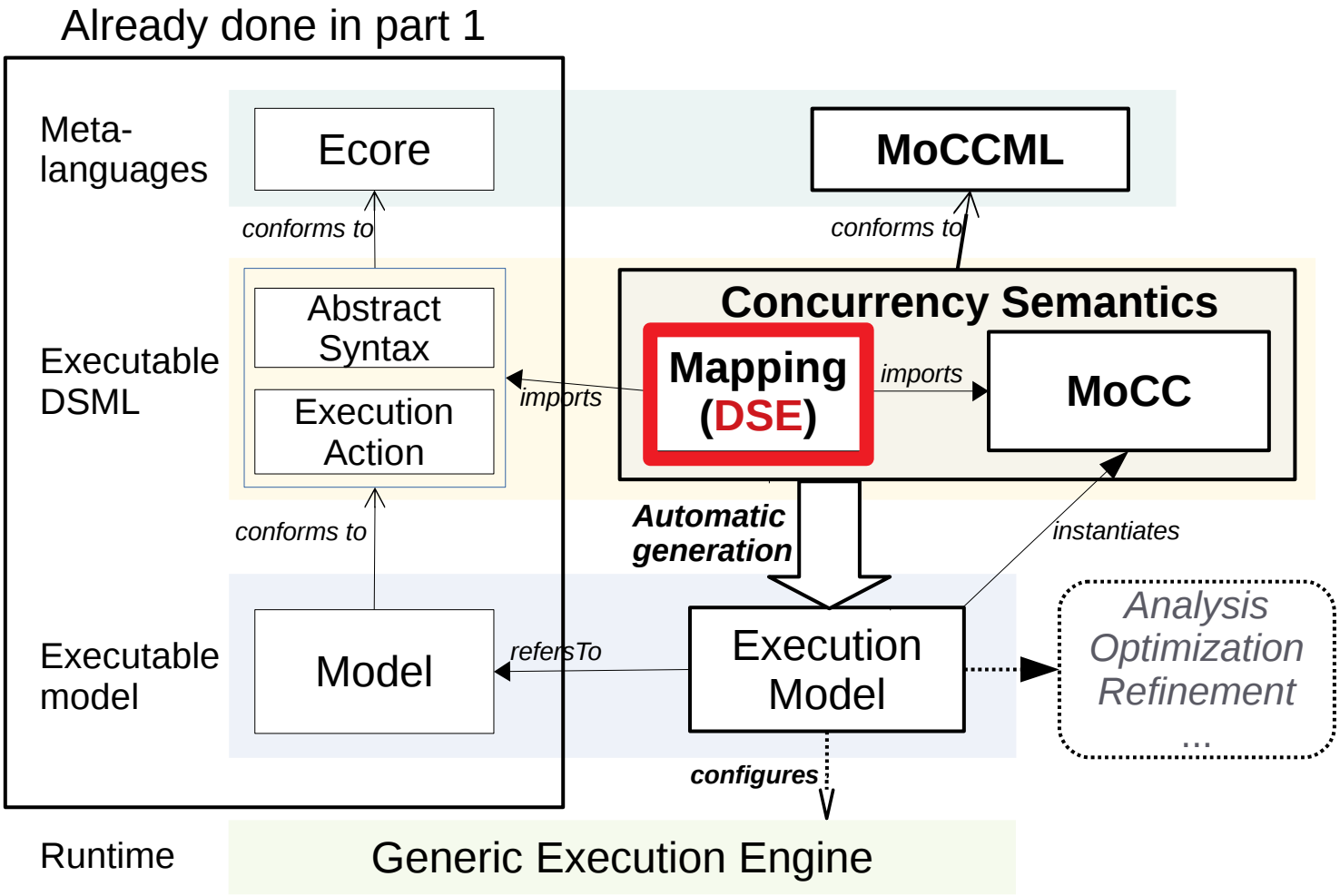
The MoCCML approach



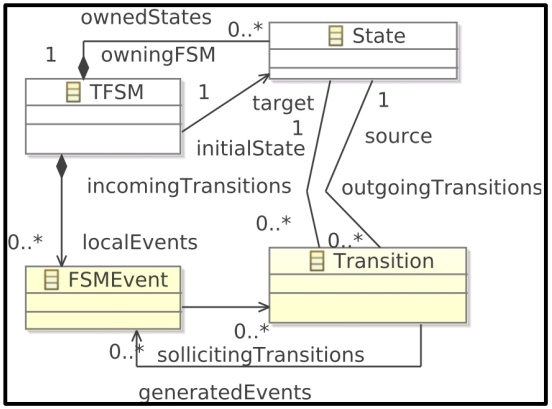
The MoCCML approach

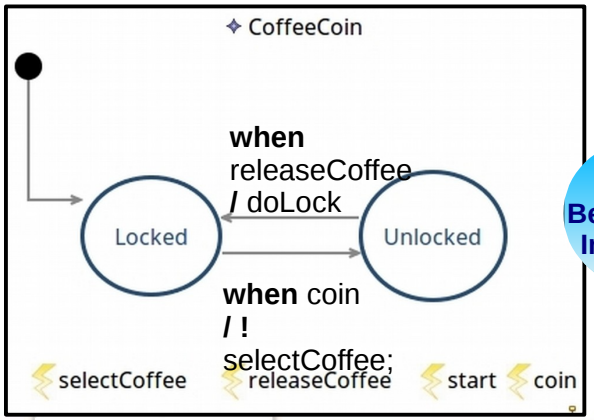


The MoCCML approach



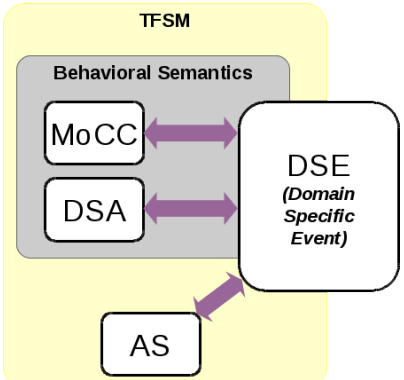
The MoCCML mapping

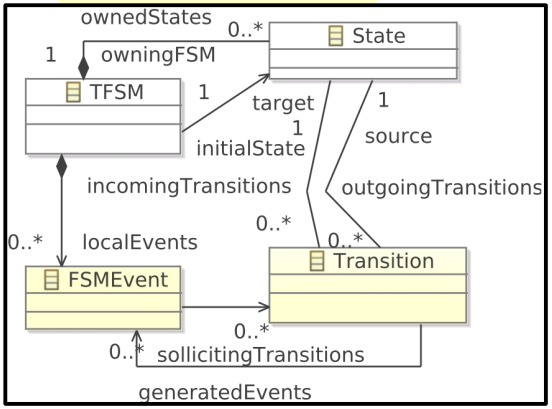


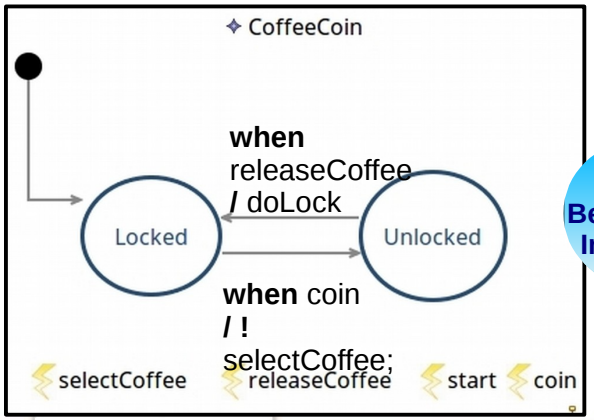


Model Behavioral Interface

The MoCCML mapping

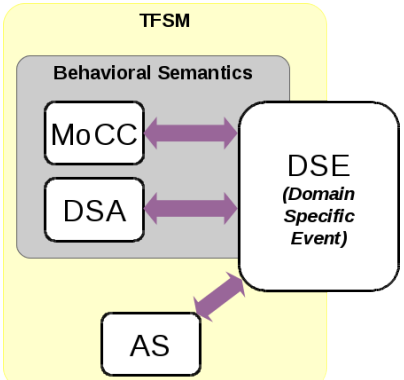


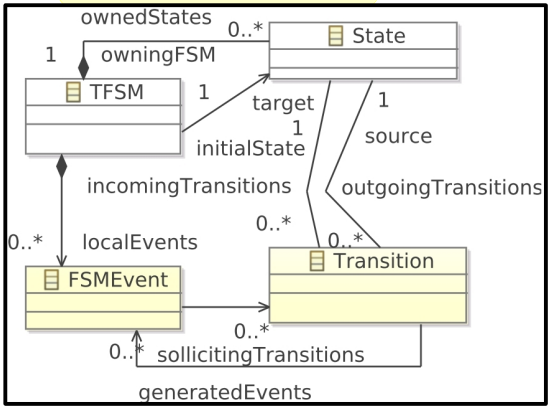




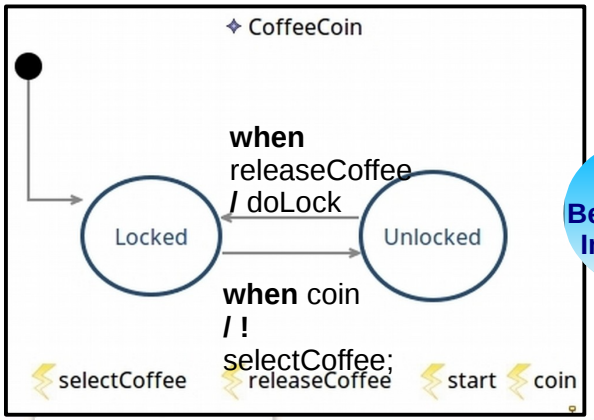
Model Behavioral Interface

The MoCCML mapping





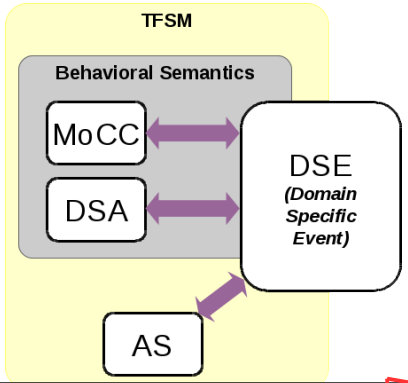

```
import 'http://org.gemoc.models17.fsm.xfsm/model/'

ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel
ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel

package model
'...'

endpackage
```

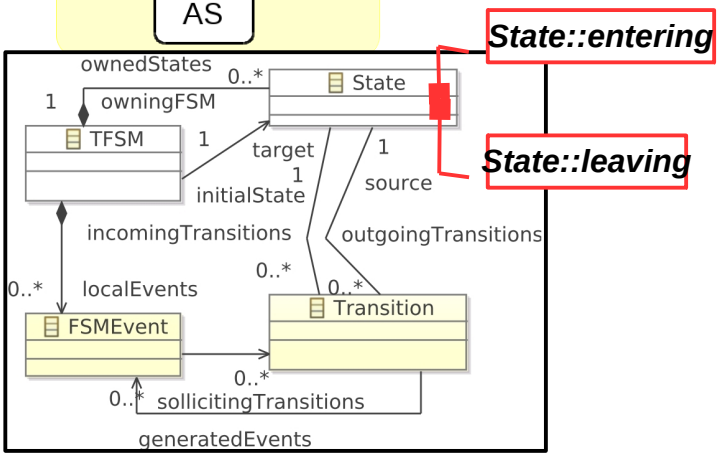


Model Behavioral Interface

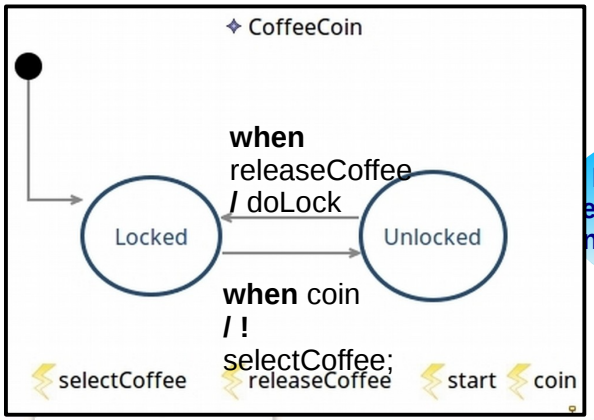
This is mainly the OCL syntax which is used...

The MoCCML mapping



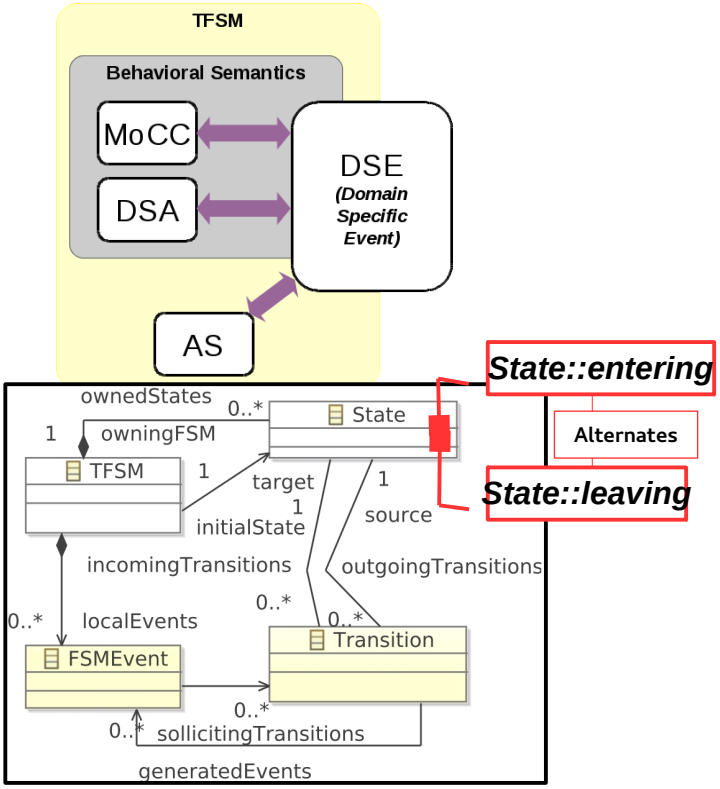


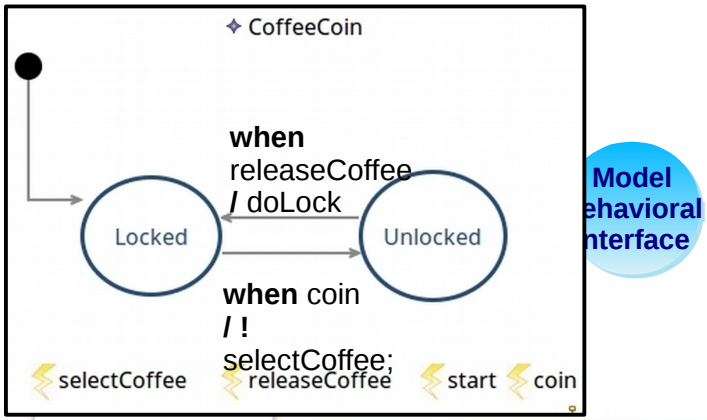

```
import 'http://org.gemoc.models17.fsm.xfsm/model/'

ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel
ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel

package model
/**
 * DSE - MoCCML mapping
 */
context State
    def: entering : Event = self
    def: leaving : Event = self

endpackage
```



Model behavioral interface

The MoCCML mapping

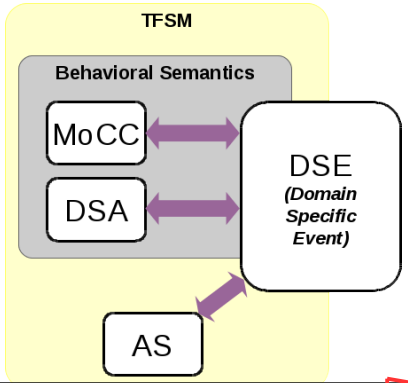


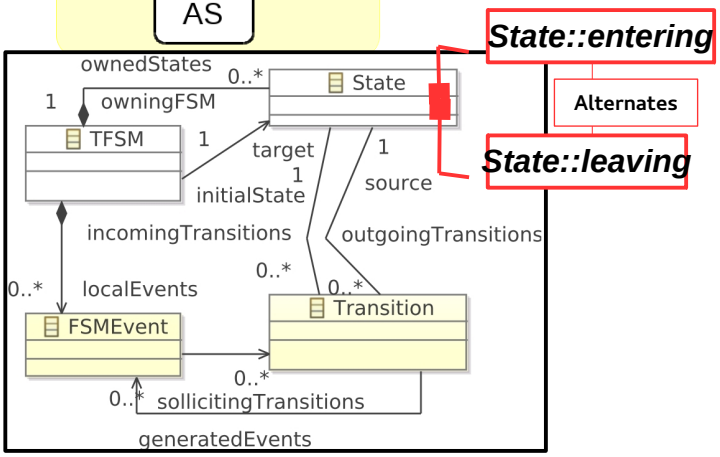


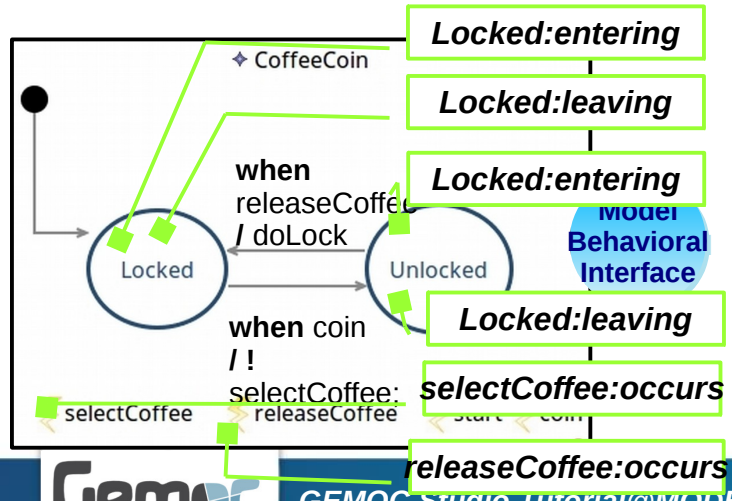
Model behavioral interface

```
import 'http://org.gemoc.models17.fsm.xfsm/model/'

ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel
ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel

package model
/**
 * DSE - MoCCML mapping
 */
context State
    def: entering : Event = self
    def: leaving : Event = self

/**
 * Constraints
 */
context State
    inv enterThenLeaveNoReEntrance:
        Relation Alternates(self.entering, leaving)

endpackage
```

The MoCCML mapping

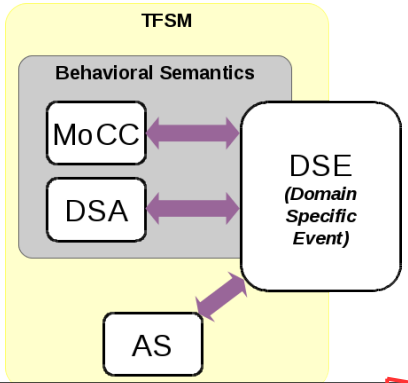


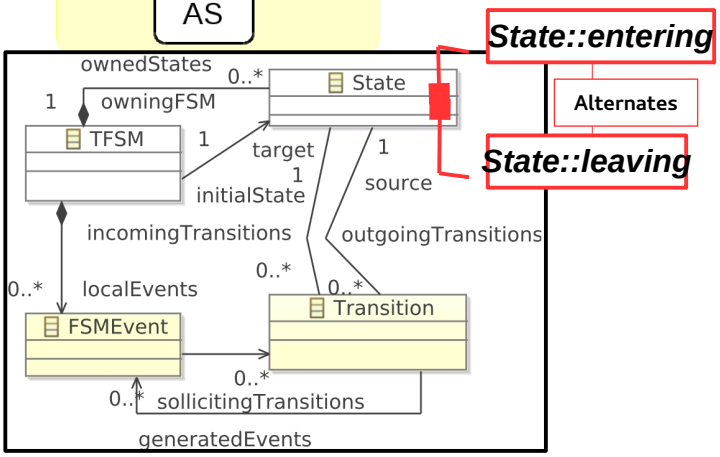




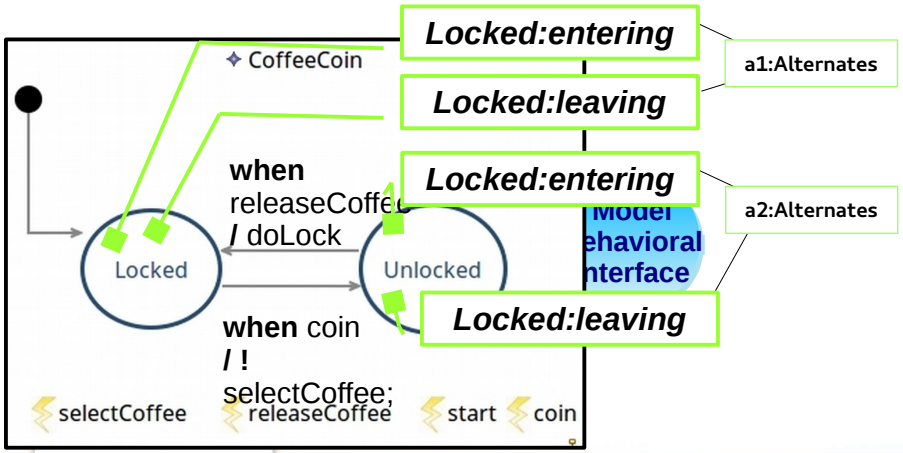

```
import 'http://org.gemoc.models17.fsm.xfsm/model/'

ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel
ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel

package model
/**
 * DSE - MoCCML mapping
 */
context State
    def: entering : Event = self
    def: leaving : Event = self

/**
 * Constraints
 */
context State
    inv enterThenLeaveNoReEntrance:
        Relation Alternates(self.entering, leaving)

endpackage
```

The MoCCML mapping



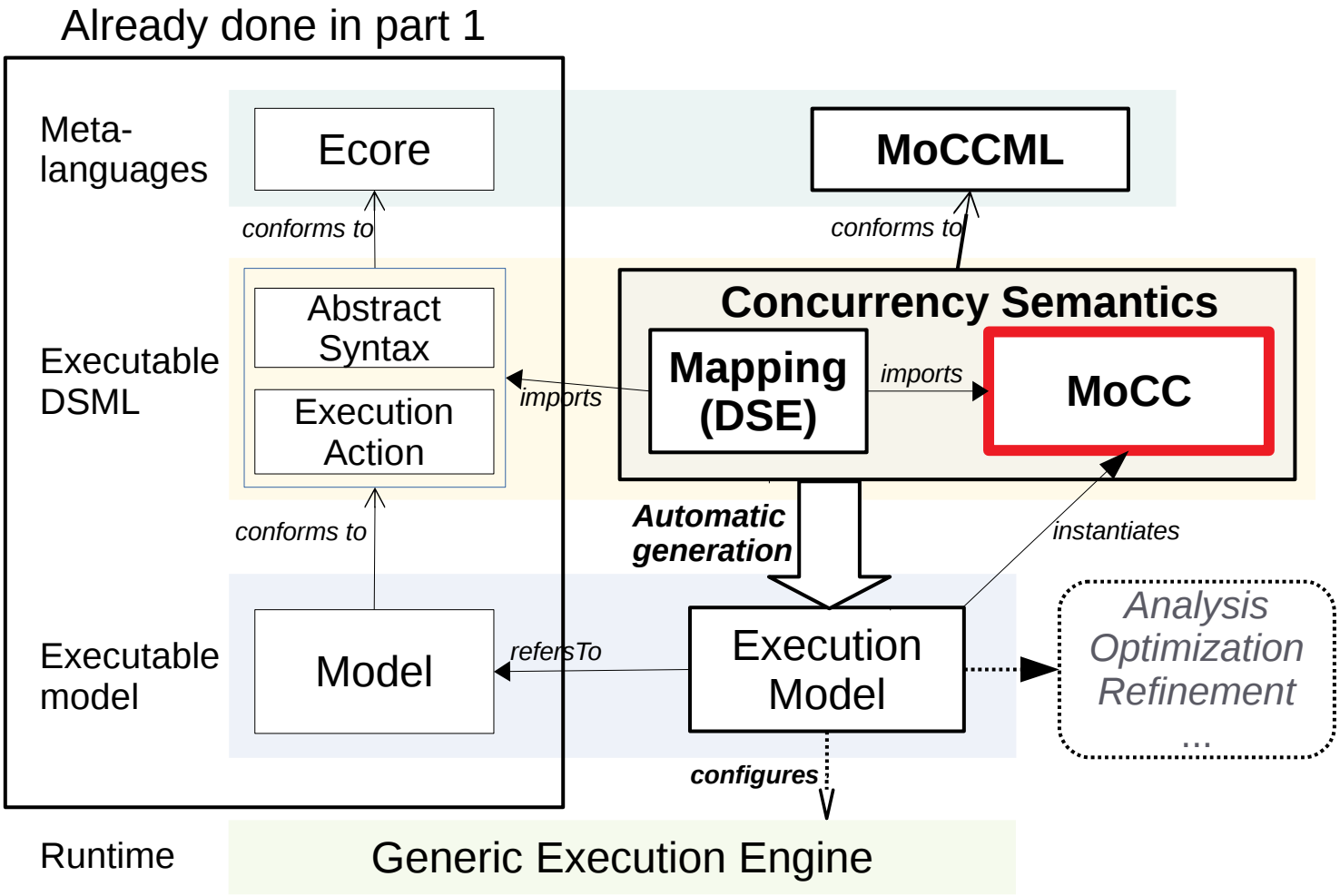



```
import 'http://org.gemoc.models17.fsm.xfsm/model/'

ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel
ECLimport "platform:/plugin/fr.inria.aoste.timesquare.ccslkernel

package model
/**
 * DSE - MoCCML mapping
 */
context State
    def: entering : Event = self
    def: leaving : Event = self

/**
 * Constraints
 */
context State
    inv enterThenLeaveNoReEntrance:
        Relation Alternates(self.entering, leaving)

endpackage
```



In green, this is the generated execution model, which is a symbolic formal representation of the scheduling state space !

The MoCCML approach



MoCCML Domain Specific Constraints

MoCCML allows for the definition of possibly complex constraints related to a specific domain :

- PeriodicTask, SporadicTask, Deadline in real time,
- Fork, Join, ActivityTriggering in fUML
- Specific Communication Protocol in Distributed Systems

It can be defined either

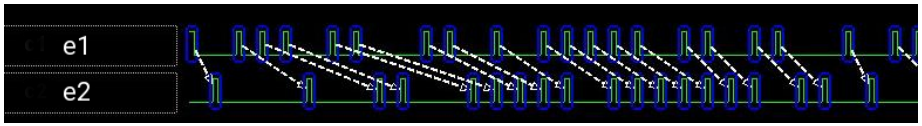
1°) based on a conjunction of primitive Event Constraints as defined in the CCSL language (*e.g., Precedes, Coincides, Excludes, DelayedFor, Alternates*)

2°) based on a MoCCML automata, i.e., a constraint automata

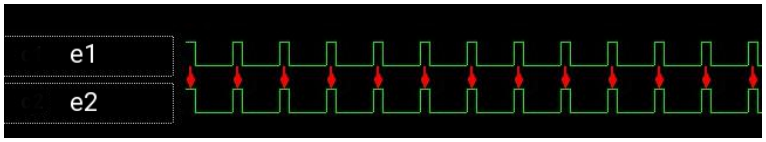
MoCCML Domain Specific Constraints

1°) based on a conjunction of primitive Event Constraints as defined in the CCSL language (*e.g., Precedes, Coincides, Excludes, DelayedFor, Alternates*)

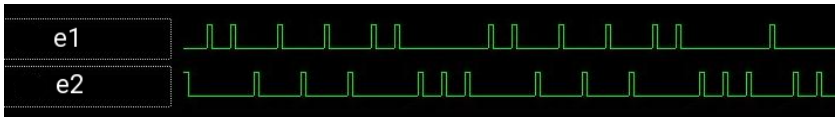
Precedes: **e1 precedes e2** means that the ith occurrence of e1 arrives before the ith of e2



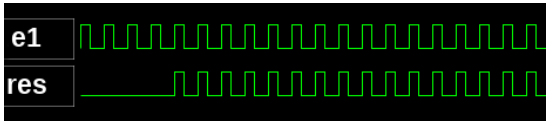
Coincides: **e1 coincides with e2** means that the ith occurrence of e1 arrives synchronously with the ith of e2



Excludes: **e1 excludes with e2** means that none of the occurrences of e1 arrives synchronously with one of e2



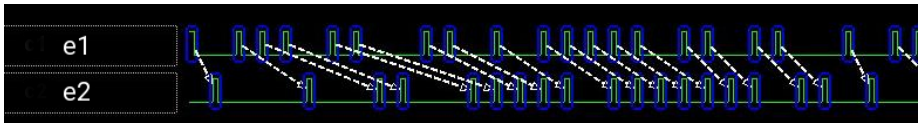
DelayedFor: **res = e1 DelayedFor N on e2** means that for each occurrences of e1 between two occurrences of e2, there is an occurrence of res after N occurrences of e2. A special case is res = e1 delayedFor N on e1. In this case the N first occurrences of e1 are removed in res.



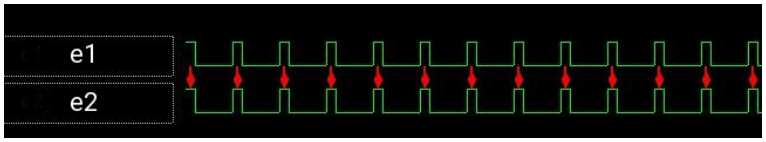
MoCCML Domain Specific Constraints

1°) based on a conjunction of primitive Event Constraints as defined in the CCSL language (*e.g., Precedes, Coincides, Excludes, DelayedFor, Alternates*)

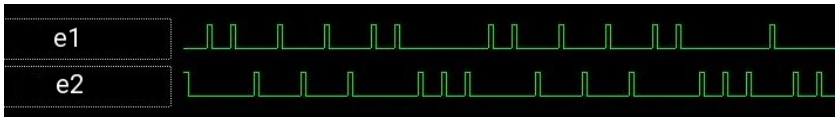
Precedes: **e1 precedes e2** means that the ith occurrence of e1 arrives before the ith of e2



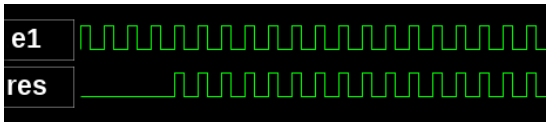
Coincides: **e1 coincides with e2** means that the ith occurrence of e1 arrives synchronously with the ith of e2

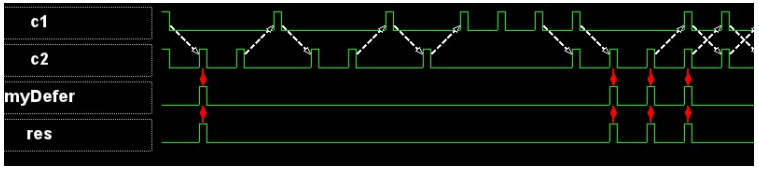


Excludes: **e1 excludes with e2** means that none of the occurrences of e1 arrives synchronously with one of e2



DelayedFor: **res = e1 DelayedFor N on e2** means that for each occurrences of e1 between two occurrences of e2, there is an occurrence of res after N occurrences of e2. A special case is res = e1 delayedFor N on e1. In this case the N first occurrences of e1 are removed in res.





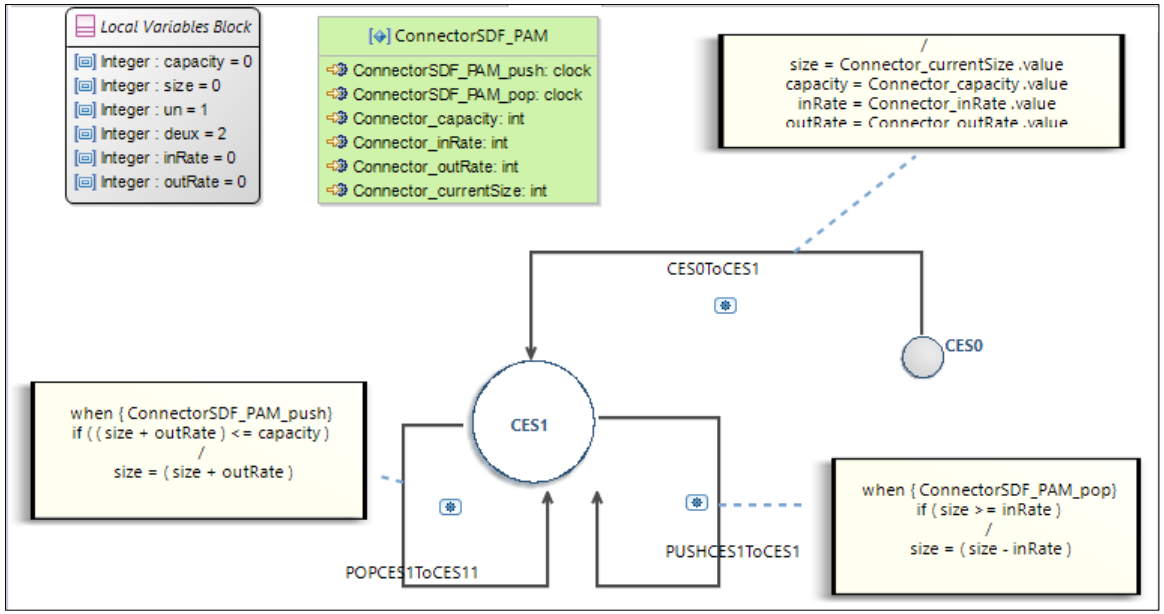
MoCCML language

MoCCML automata intuitive semantics

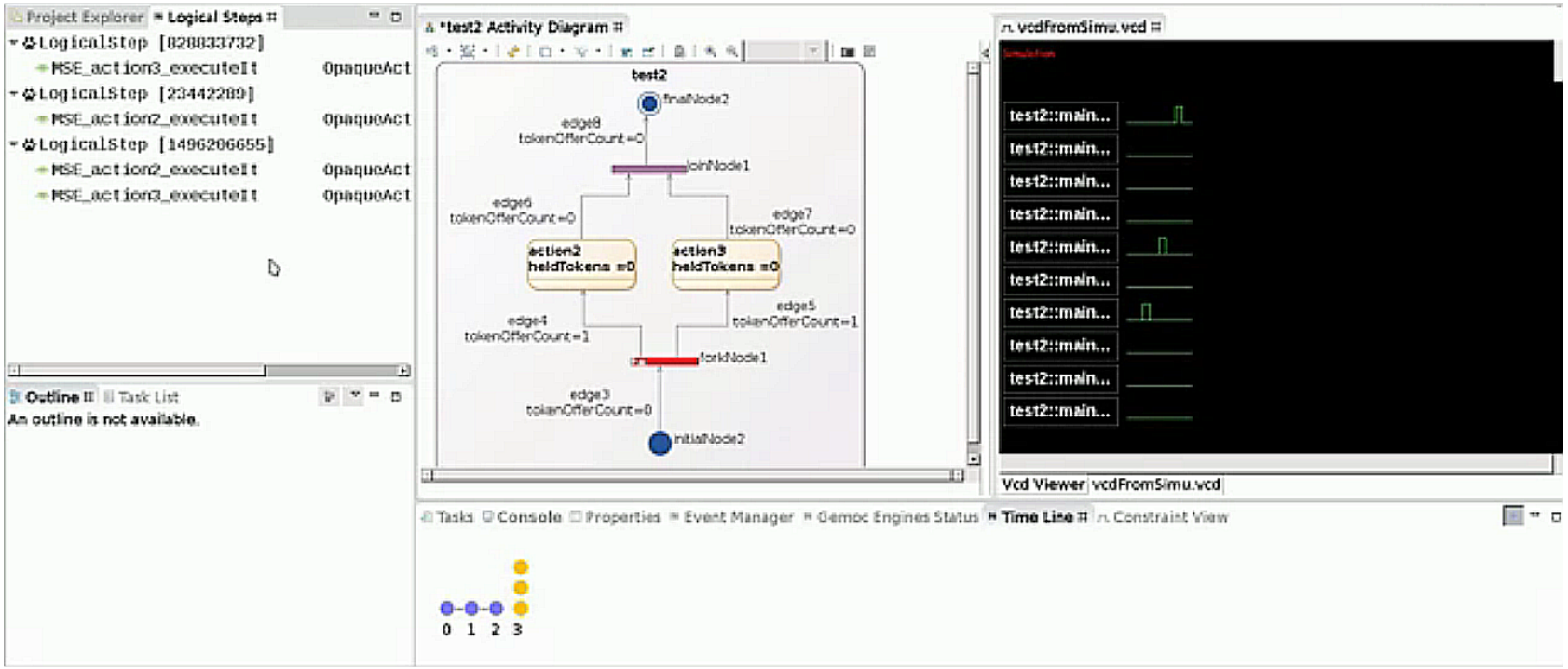
- Clocks as transition triggers
- Condition expressions on integer variables
- Assignment and arithmetic operators on variables

Operational Semantics of the Model of Concurrency and Communication Language :

https://hal.inria.fr/hal-01060601v2



Activity Diagram Debugger



Benoit Combemale, Julien Deantoni, Matias Vara Larsen, Frédéric Mallet, Olivier Barais, Benoit Baudry, Robert France, "Reifying Concurrency for Executable Metamodeling" In Software Language Engineering (SLE), 2013

Hands On

Program outline (see README)

- **Language workbench: create your xFSM DSL**
  - Changing the language specification from sequential to concurrent
  - Creation of the DSE and MoCCML mapping project
  - Modification of the ECL file
  - Create Domain Specific MoCC constraints

Wrap-up and discussion

- Today you have
  - executed models of the communicating FSM language
  - Completed the **operational semantics** by using **Kermeta**
  - extended its concrete **graphical syntax for animation** using **Sirius**
  - Added a explicit and **formal concurrency model** to your semantics by using **MoCCML**
  - Saw hox to coordinate **heterogeneous executable modeling languages** to support concurrent model execution by using **BCOoL**
  - Had fun ?