



BCOOL

The *B*ehavioral *C*oordination *O*perator *L*anguage

MPM4CPS

27th of April 2018

Julien Deantoni

Universite Cote d'Azur,

CNRS I3S, INRIA KAIROS

Julien.deantoni@polytech.unice.fr

Modeling the behavioral semantics of languages & their coordination

MPM4CPS

27th of April 2018

Julien Deantoni

Universite Cote d'Azur,

CNRS I3S, INRIA KAIROS

Julien.deantoni@polytech.unice.fr

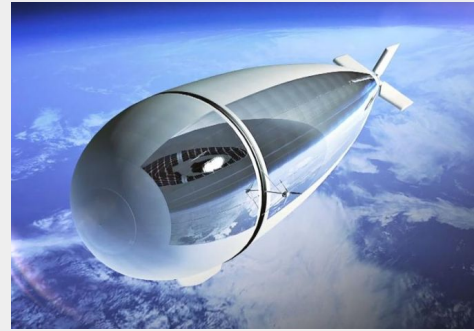
System

The term "system" comes from the *Latin* word *systema*, in turn from *Greek* σύστημα : "a whole made of several parts or members".

wikipedia



Airbus 390



Thales Alenia Space: stratobus and satellites

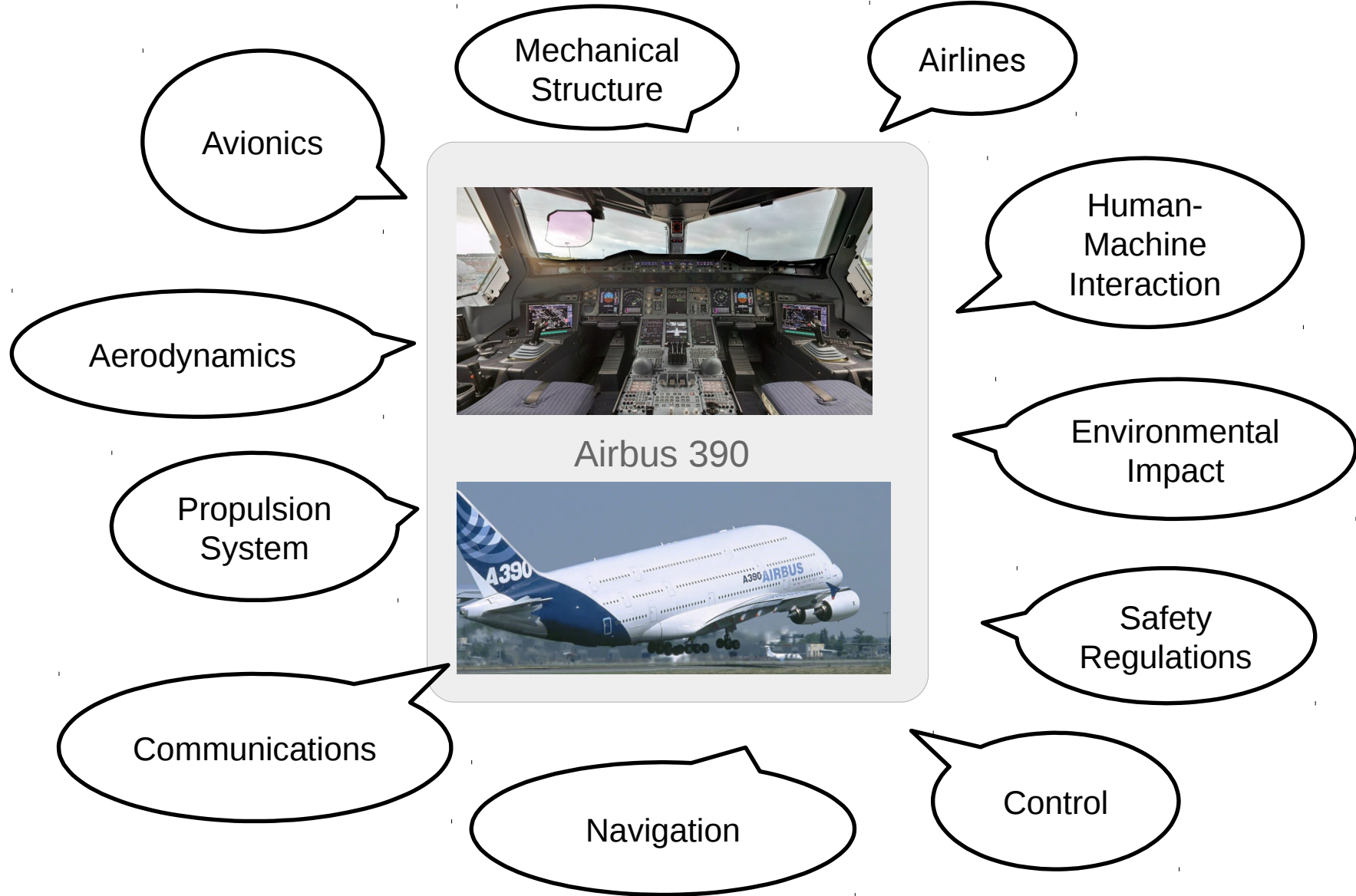
Many stakeholders are needed to develop such systems

Renault Autonomous Car



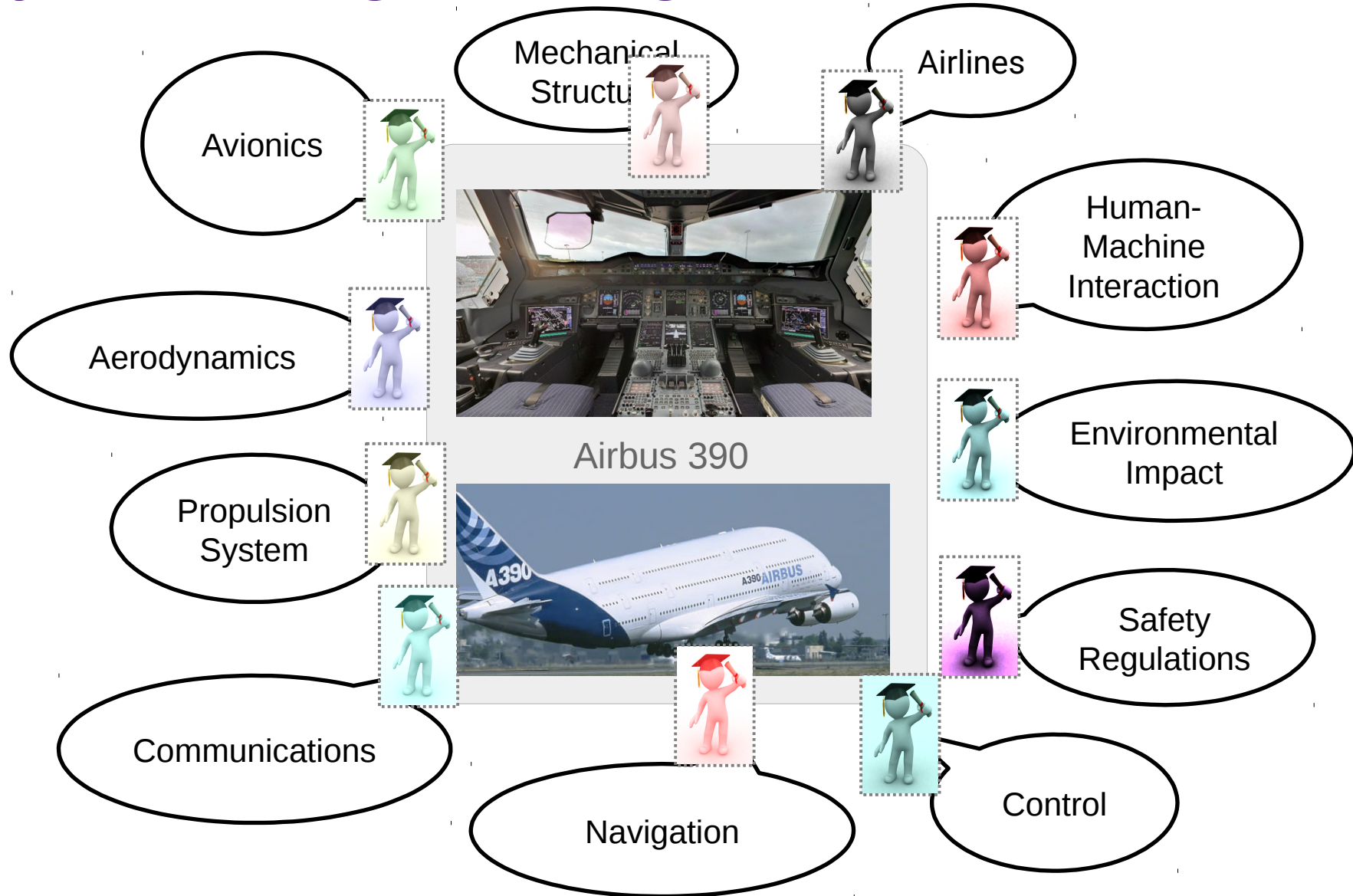
Safran Nacelle

System Engineering



Several concerns for a single system

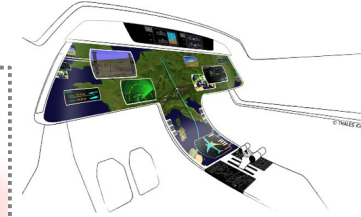
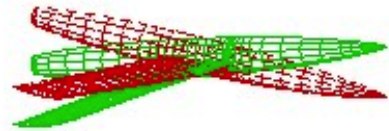
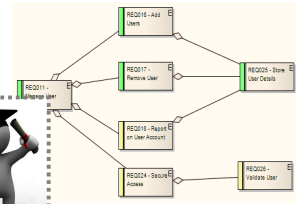
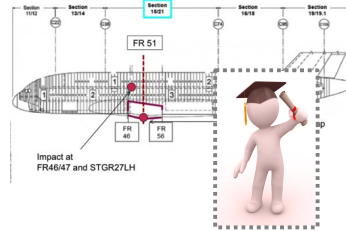
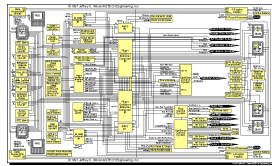
System Engineering



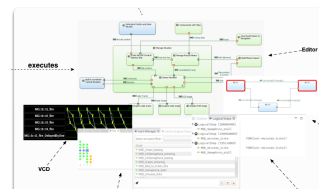
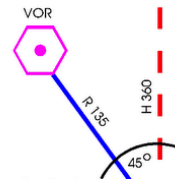
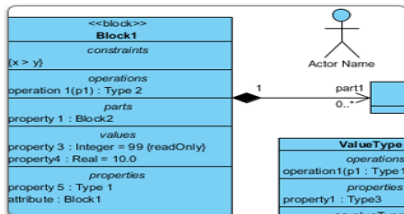
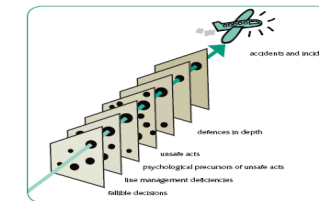
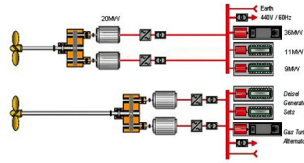
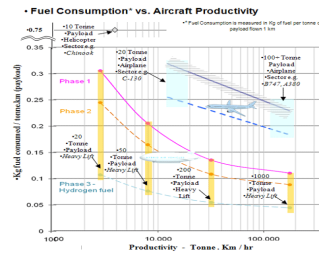
Several concerns for a single system

At least one expert by concern

System Engineering



Airbus 390

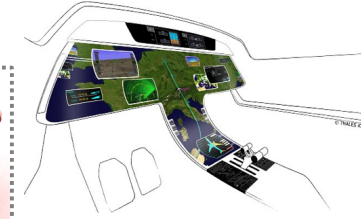
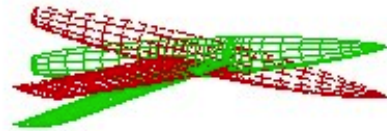
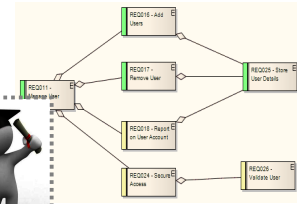
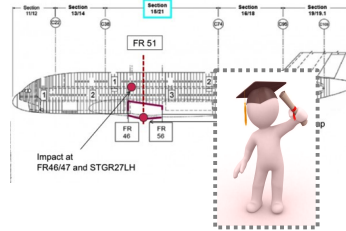
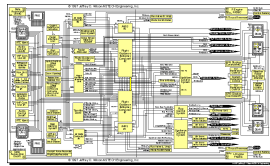


Several concerns for a single system

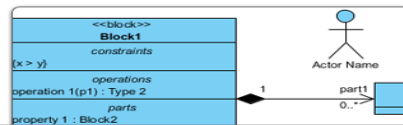
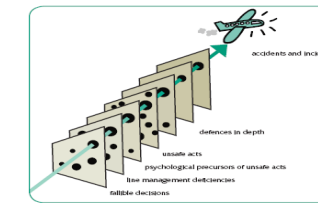
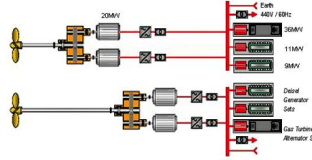
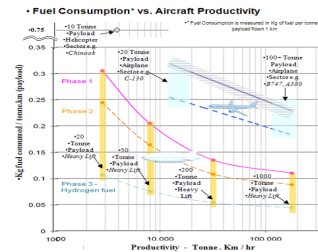
At least one expert by concern

At least one "model" by concern (expressed in a DSML)

System Engineering

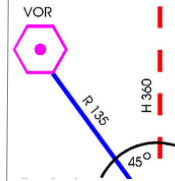


Airbus 390



a system is represented by a set of functional and non functional heterogeneous models

Several concerns for a single system

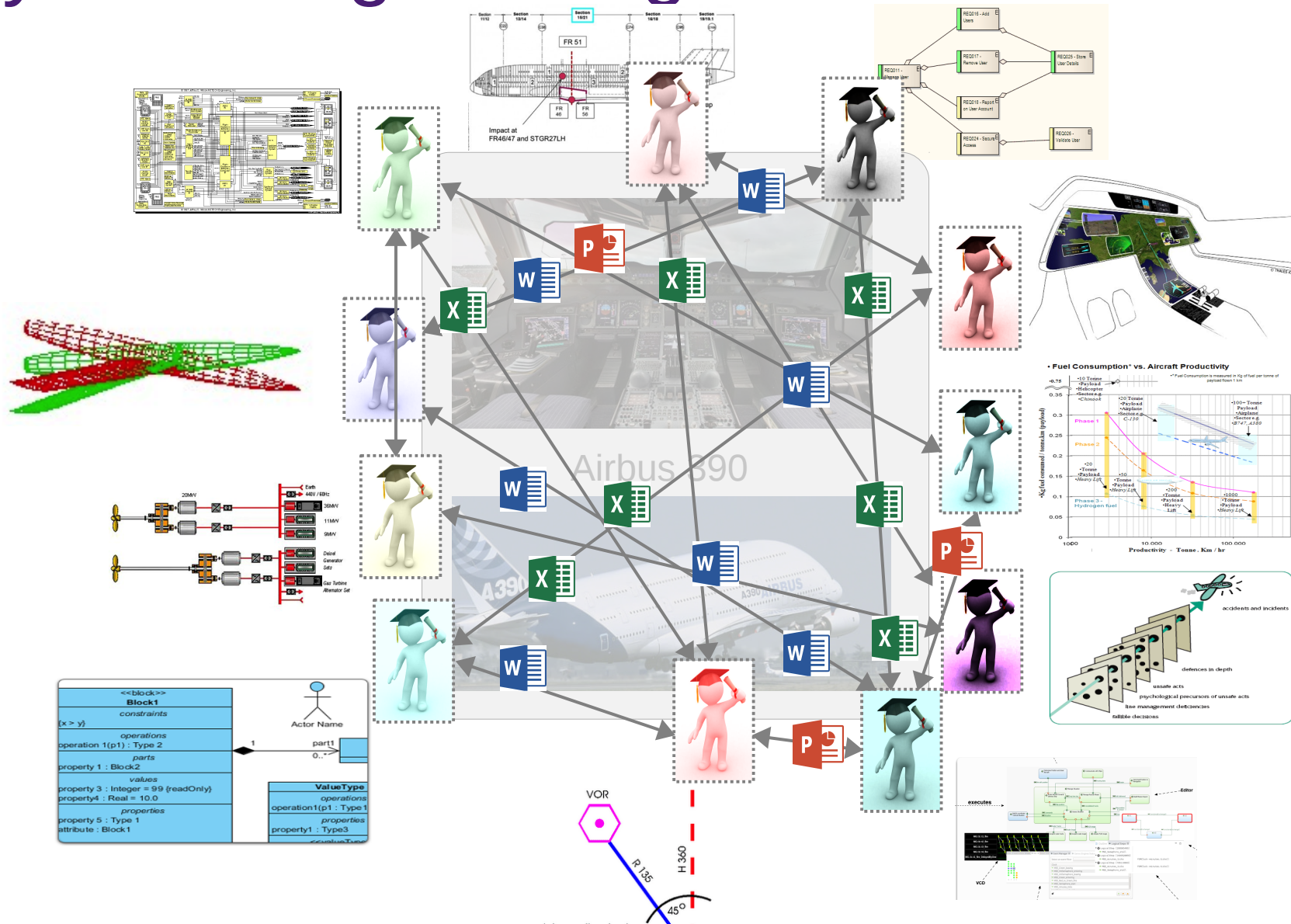


At least one expert by concern

a Model is an Abstraction of a specific Concern/View of a system for a given Purpose

At least one "model" by concern (expressed in a DSML)

System Engineering

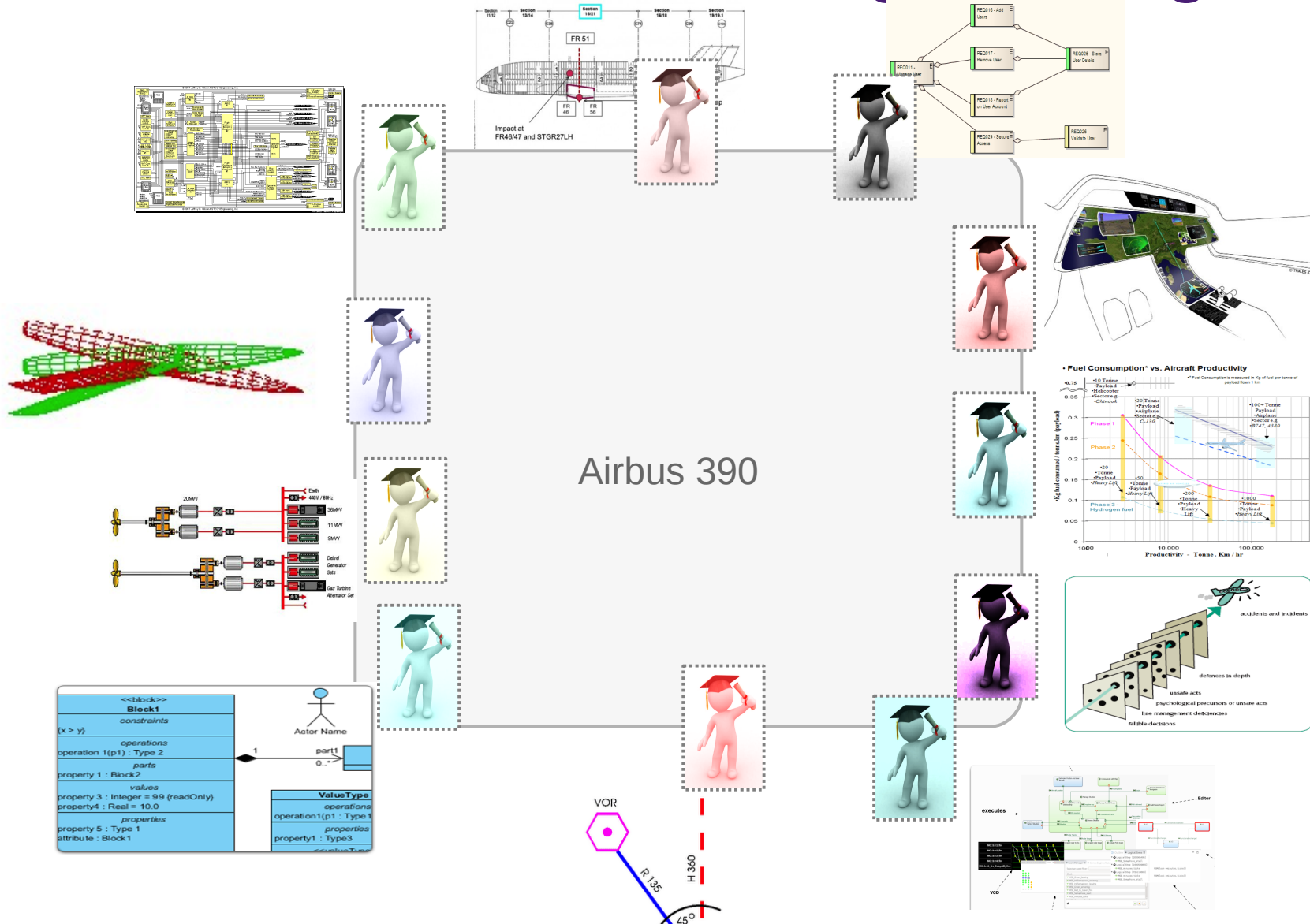


Several concerns for a single system

At least one expert by concern

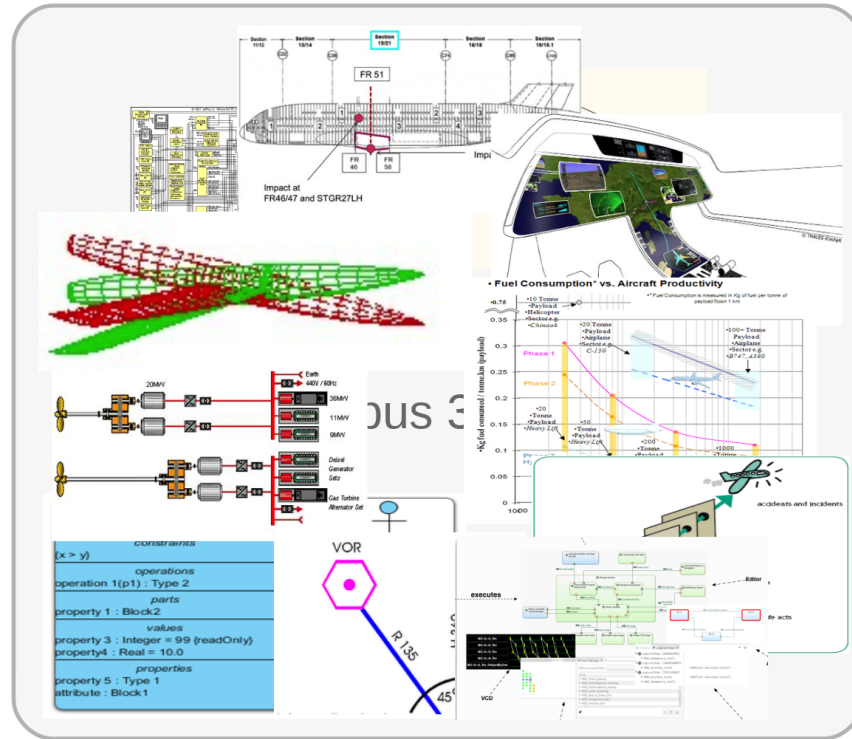
At least one "model" by concern (expressed in a DSML)

Model Based System Engineering



Model Based System Engineering specifies in a model the correspondences between models from the different concerns, all along the product life cycle.

Model Based System Engineering



Model Based System Engineering specifies in a model the correspondences between models from the different concerns, all along the product life cycle.


Model Based System Engineering

Lots of interesting challenges here...

CONSTRAINTS	
operations	
operation 1(p1) : Type 2	
parts	
property 1 : Block2	
values	
property 3 : Integer = 99 (readOnly)	
property4 : Real = 10.0	
properties	
property 5 : Type 1	
attribute : Block 1	

Model Based System Engineering specifies in a model the correspondences between models from the different concerns, all along the product life cycle.

Model Based System Engineering



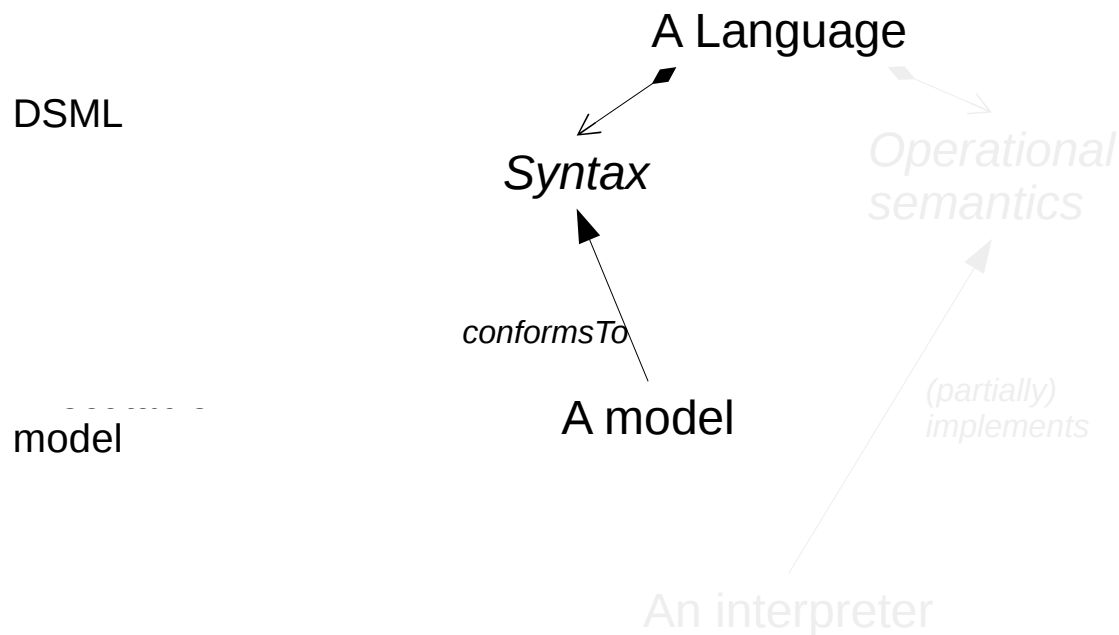
**Lots of interesting challenges here...
→ we focus on the formal reasoning about emerging behaviors**

CONSTRAINTS	
operations	
operation 1(p1) : Type 2	
parts	
property 1 : Block2	
values	
property 3 : Integer = 99 (readOnly)	
property4 : Real = 10.0	
properties	
property 5 : Type 1	
attribute : Block 1	

Model Based System Engineering specifies in a model the correspondences between models from the different concerns, all along the product life cycle.

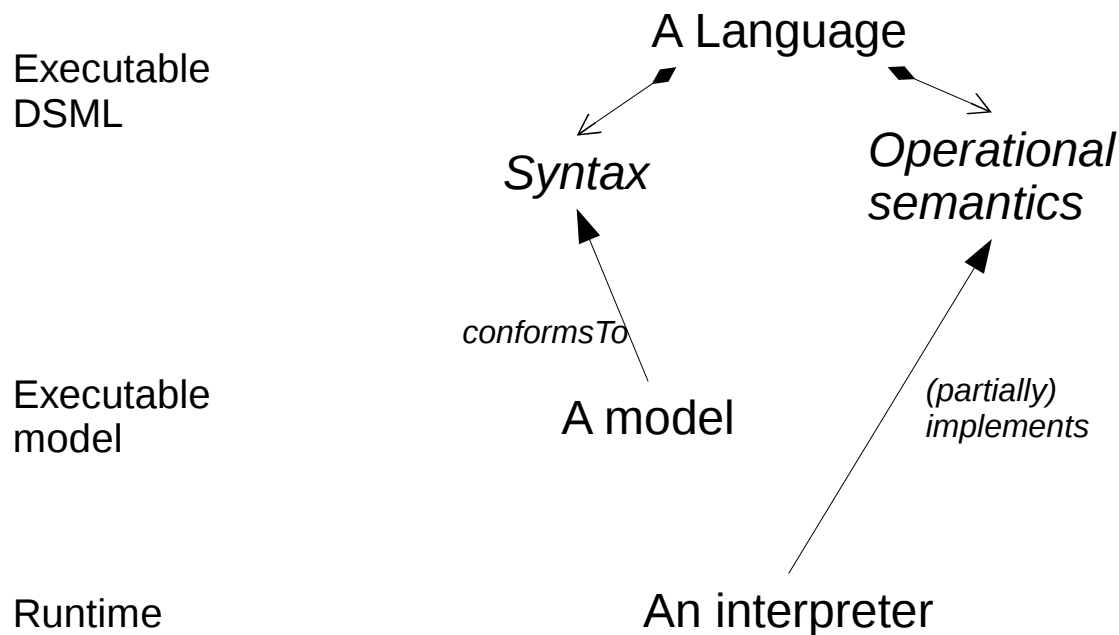
Context

- We consider models that can be interpreted according to their (concurrent and timed) operational semantics
- We do not want to implement all the tooling for each new language



Context

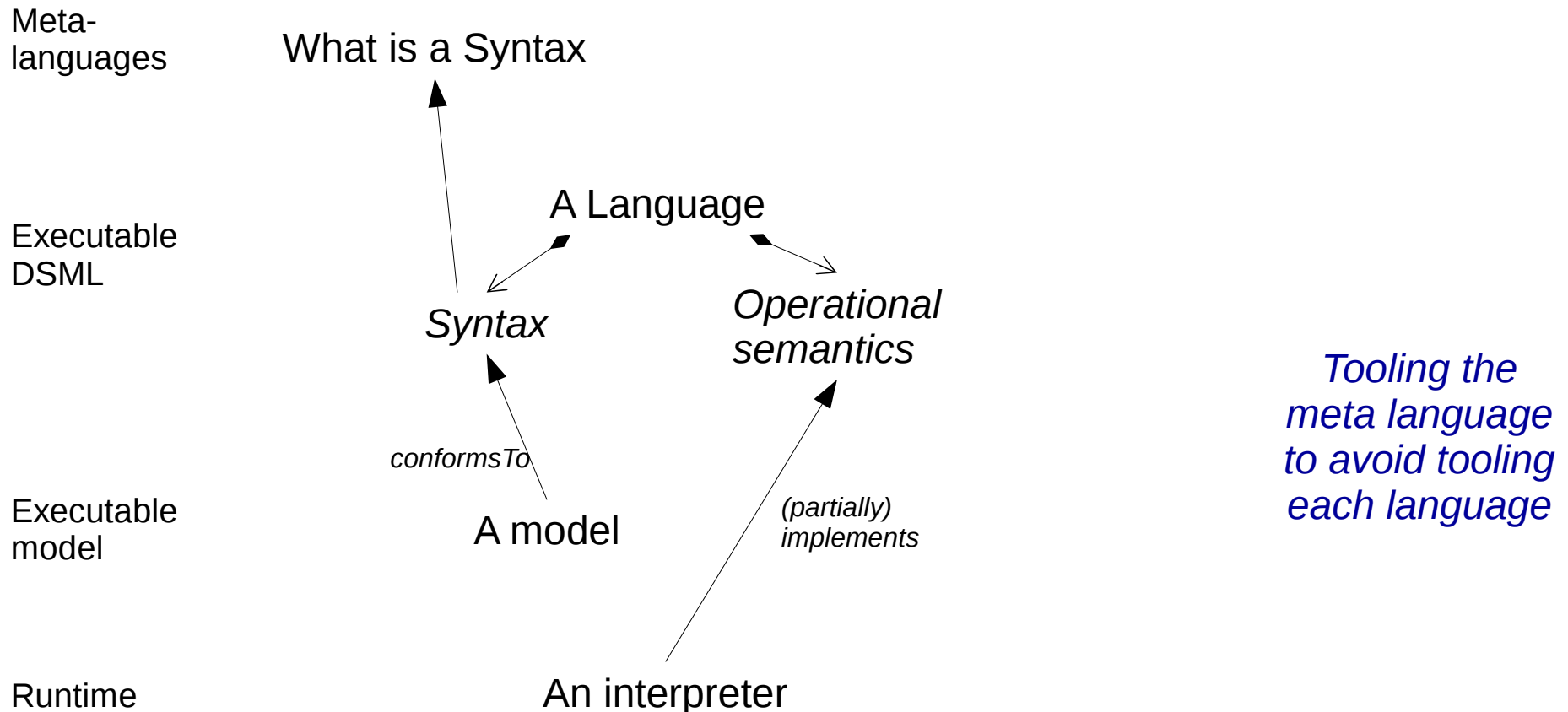
- We consider models that can be interpreted according to their (concurrent and timed) operational semantics
- We do not want to implement all the tooling for each new language



We need to make the operational semantics explicit... and as formal as possible

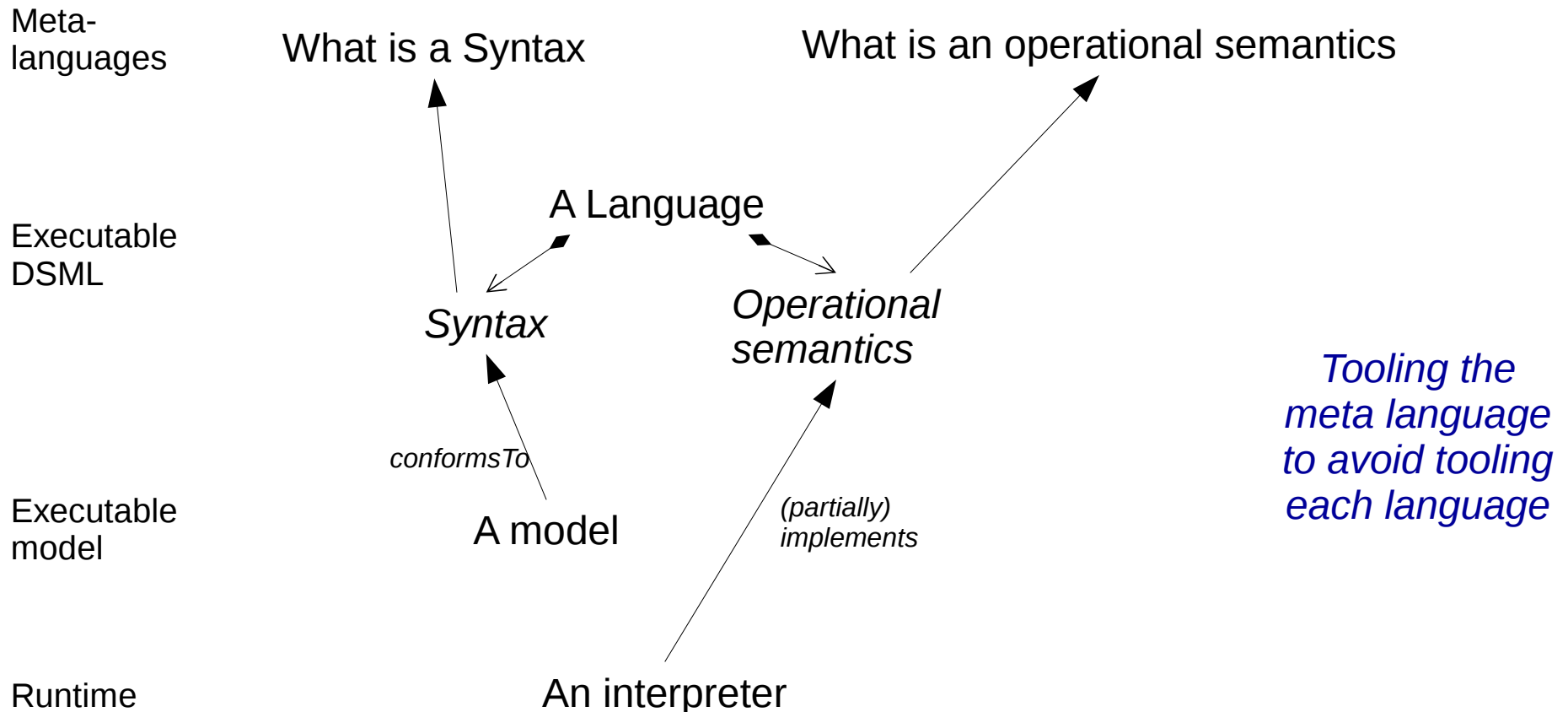
Context

- We consider models that can be interpreted according to their (concurrent and timed) operational semantics
- We do not want to implement all the tooling for each new language



Context

- We consider models that can be interpreted according to their (concurrent and timed) operational semantics
- We do not want to implement all the tooling for each new language



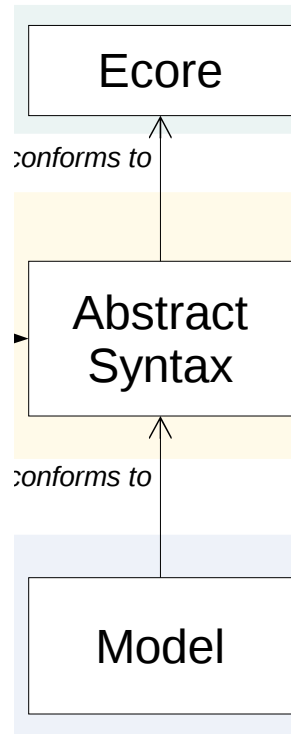
The GEMOC approach

Meta-languages

Executable DSML

Executable model

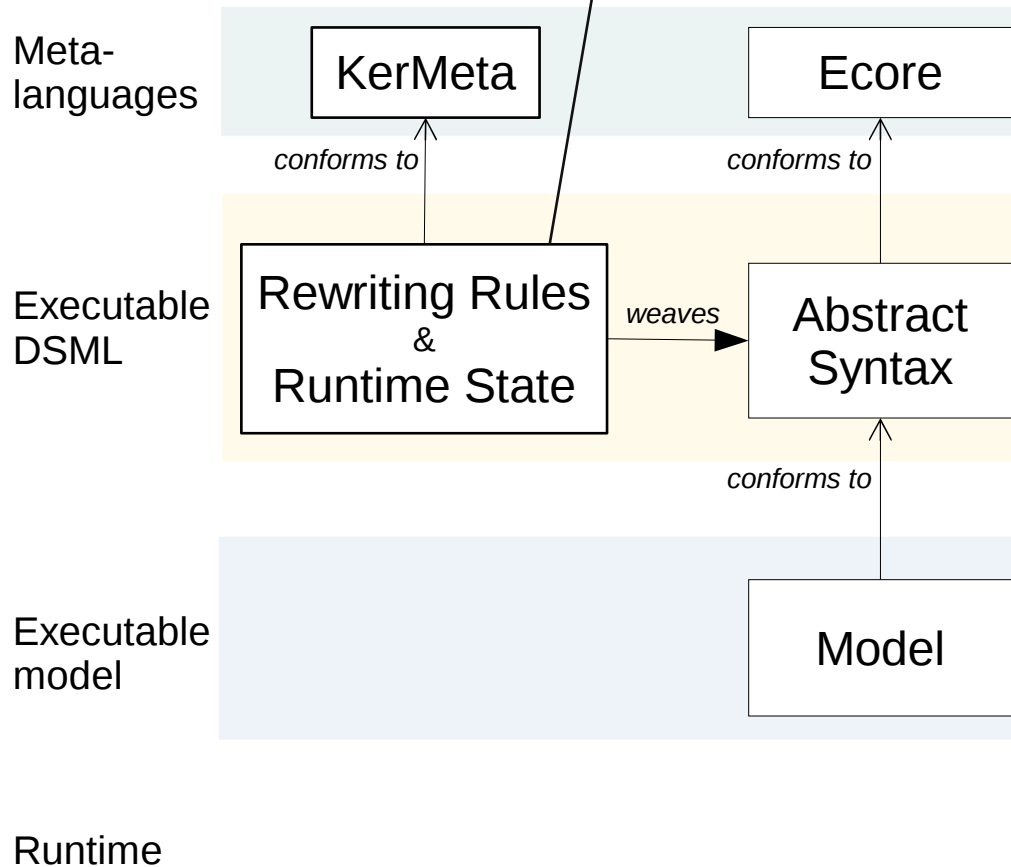
Runtime



The GEMOC approach

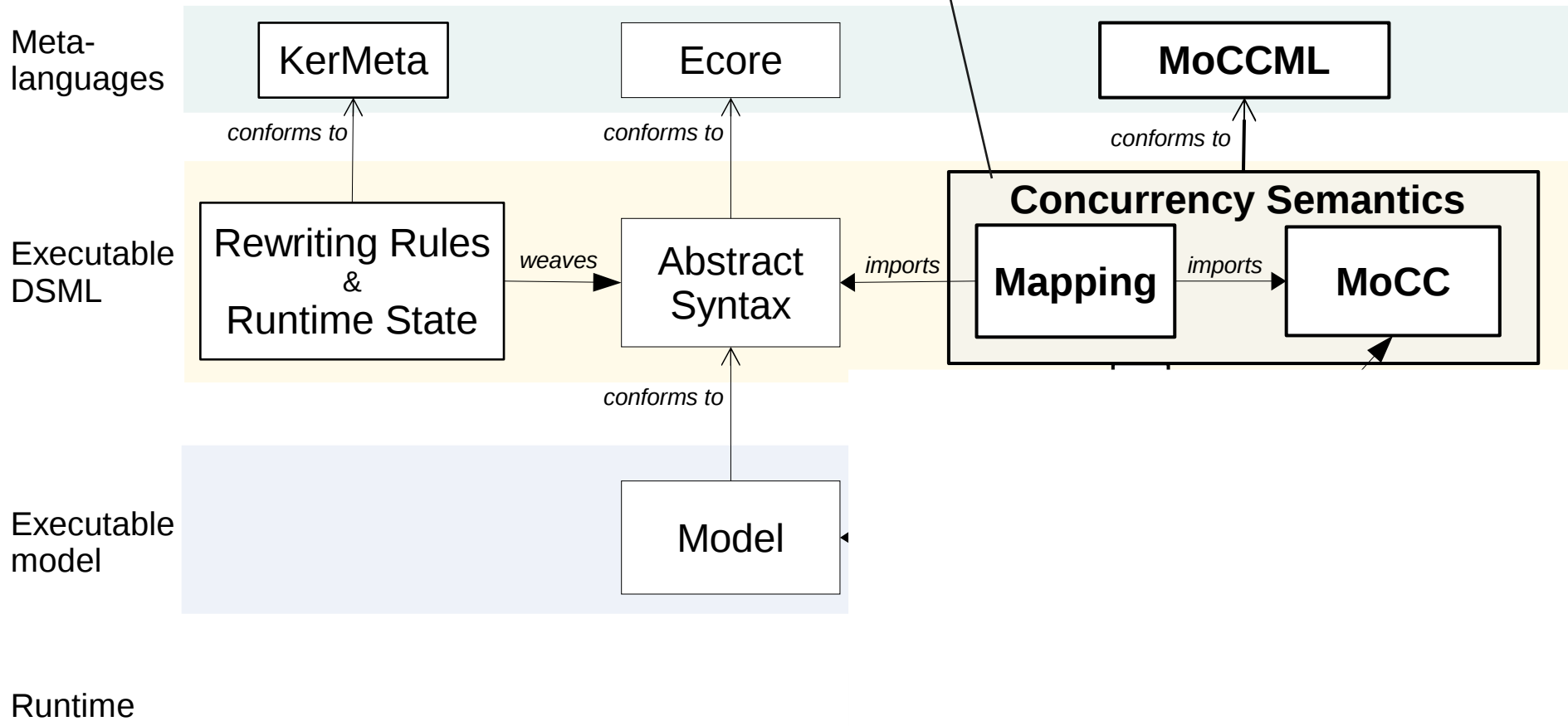
Two strongly linked parts:

- the data representing the runtime state of the model.
- The actions specifies how the model state is evolving

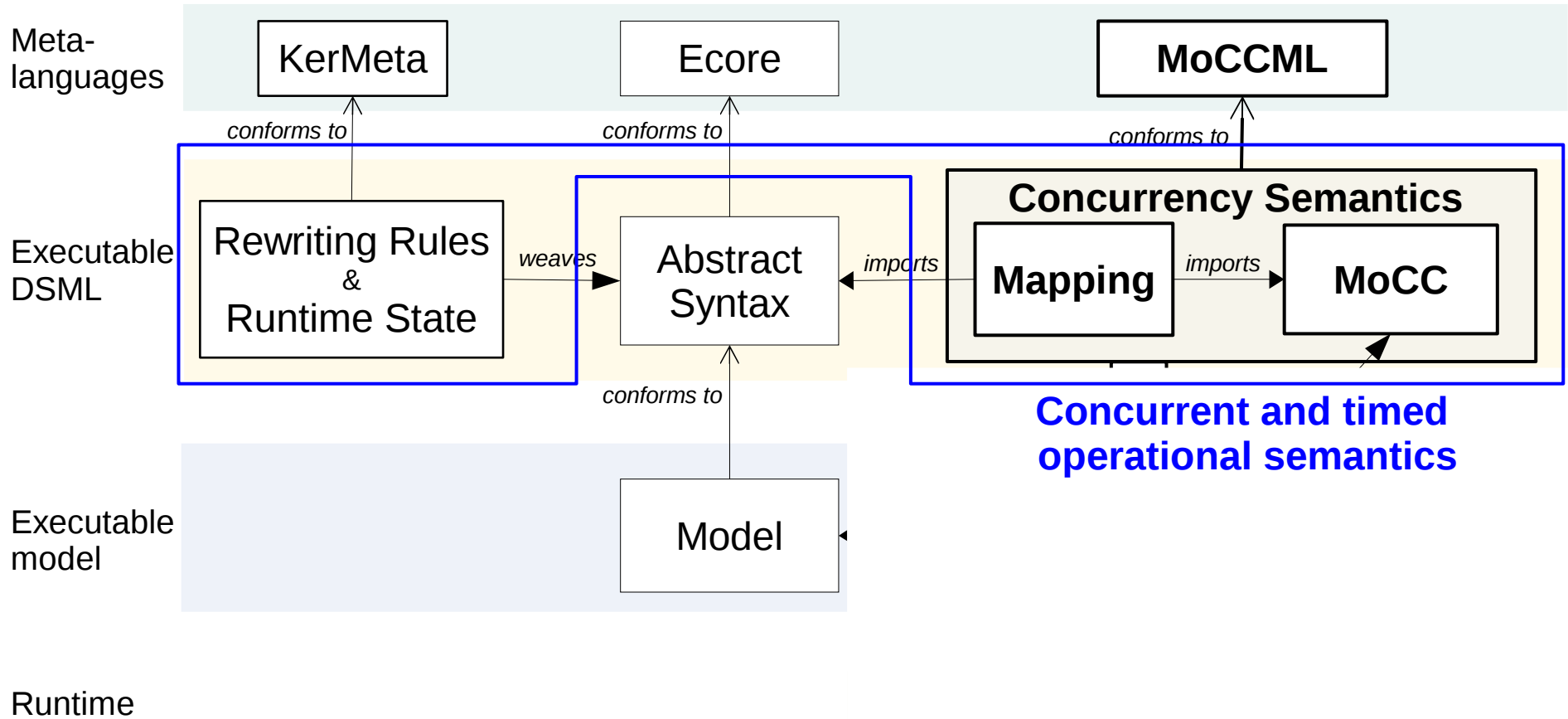


The GEMOC approach

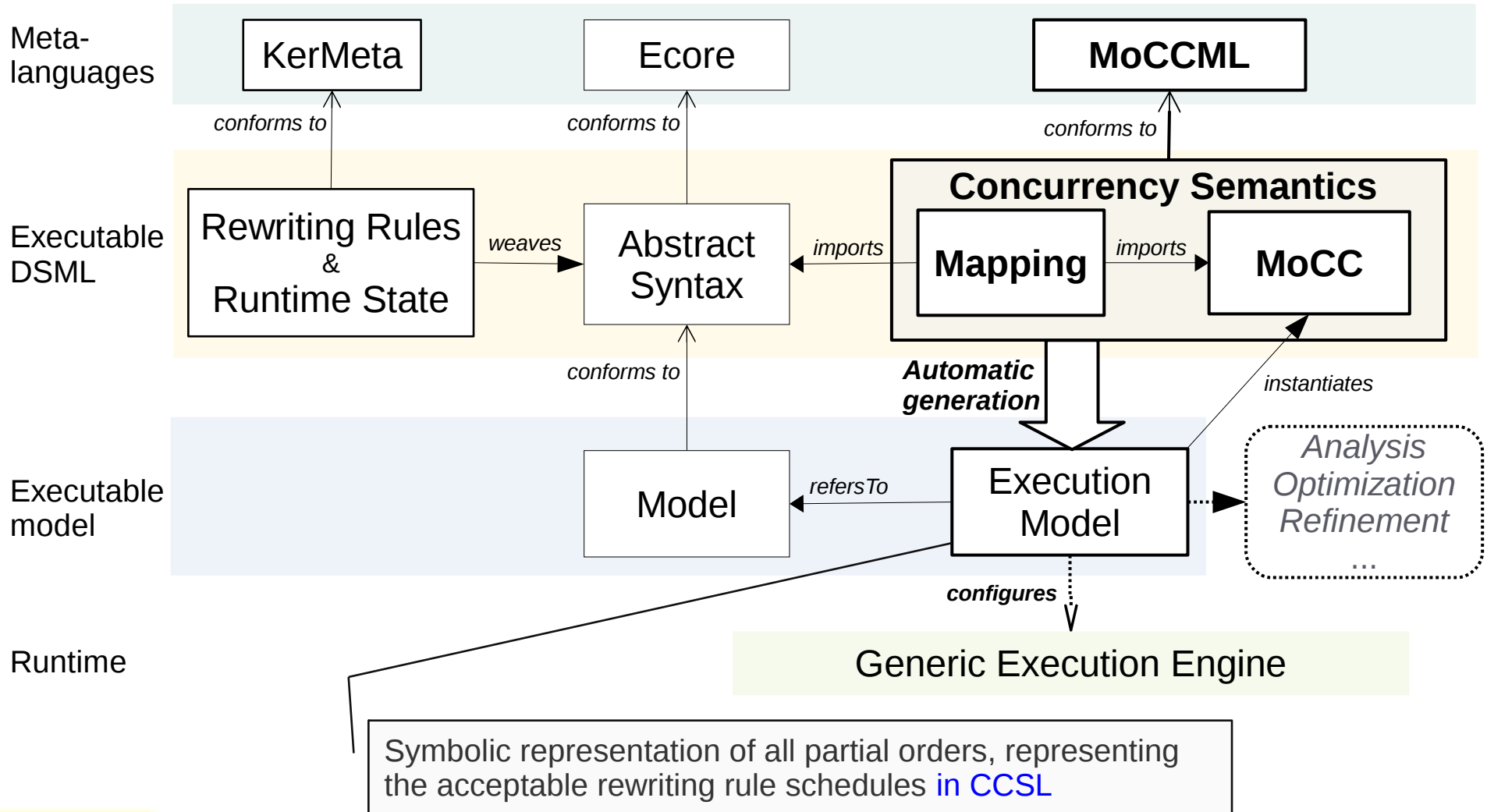
It Specifies **when** the rewriting rules that make the model evolving are called.
 It models the (possibly timed) causalities and synchronizations between the rewriting rules



The GEMOC approach

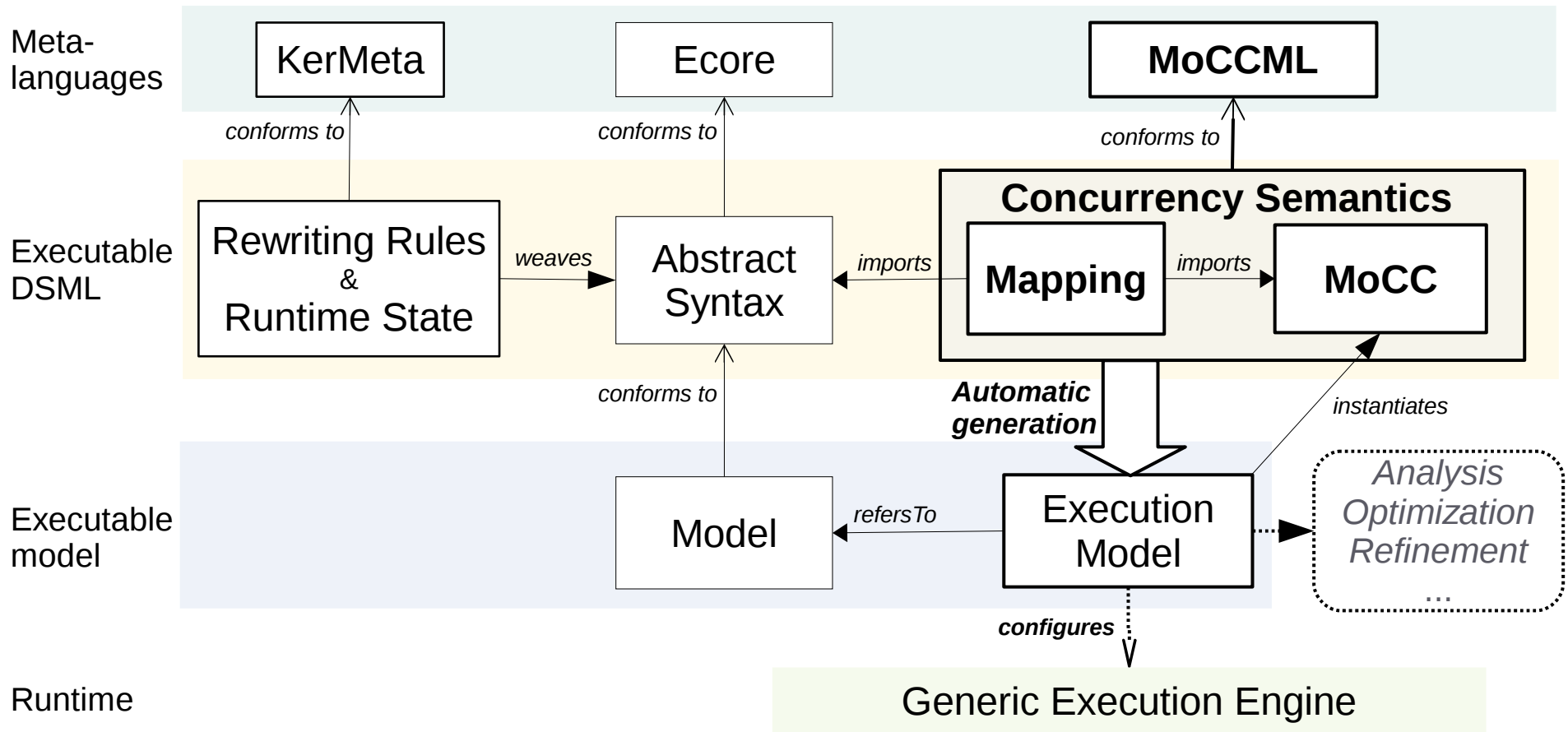


The GEMOC approach

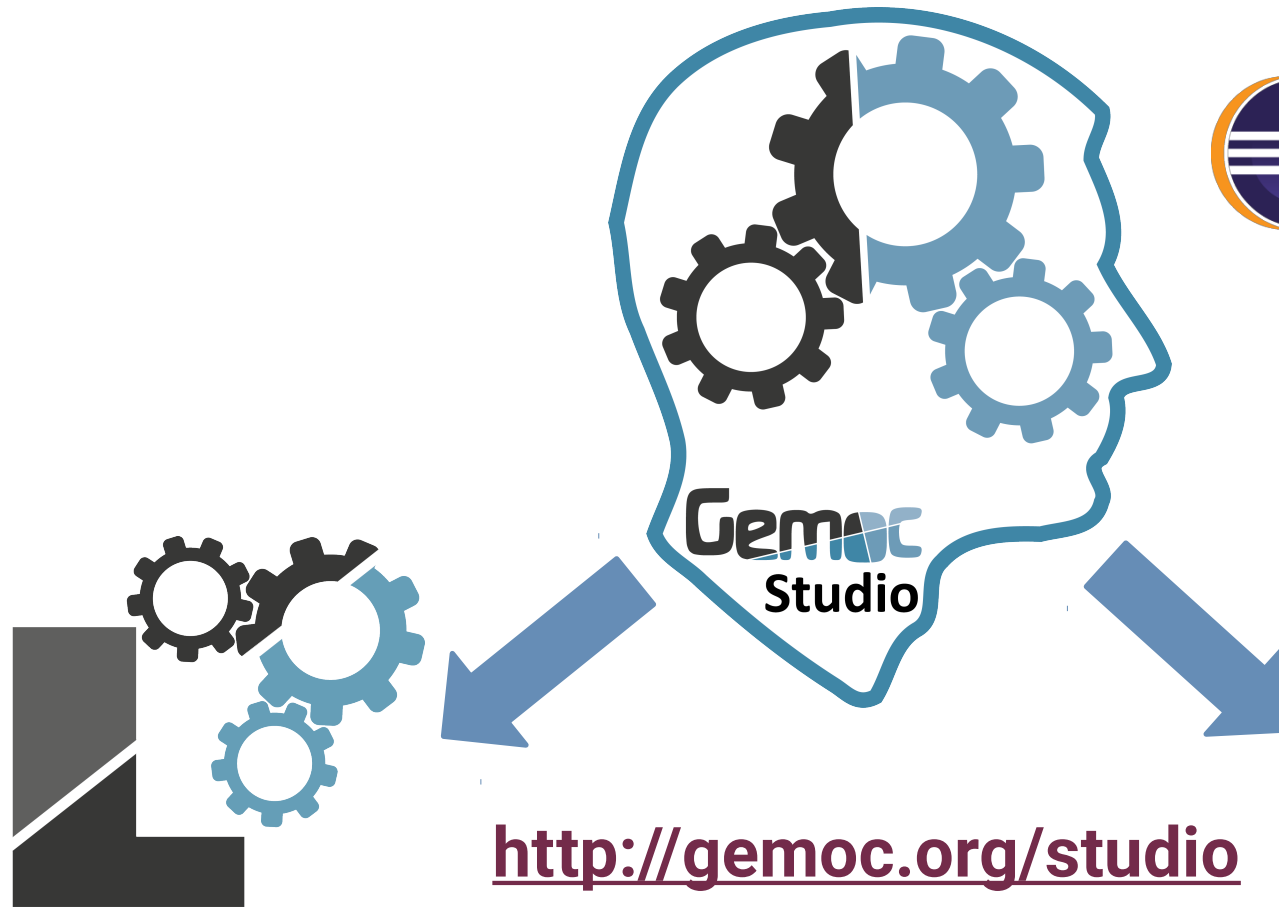


The GEMOC approach

+ graphical concrete syntax in Sirius, which uses a meta-language as well



The GEMOC Studio



eclipse

Research Consortium

<http://eclipse.org/gemoc>



Modeling Workbench

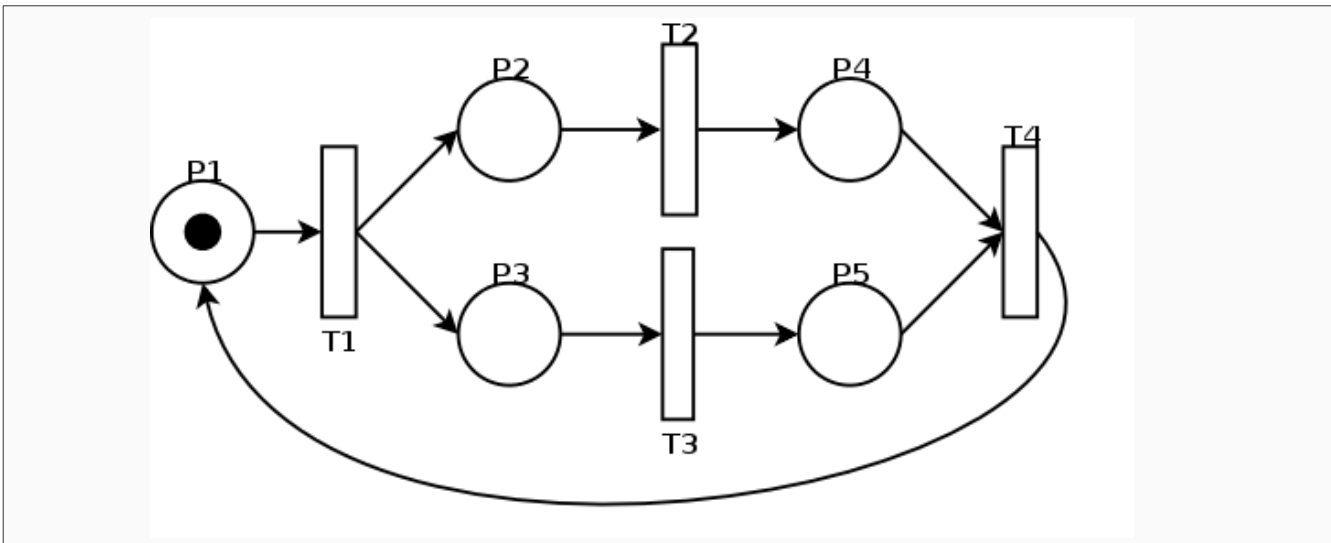
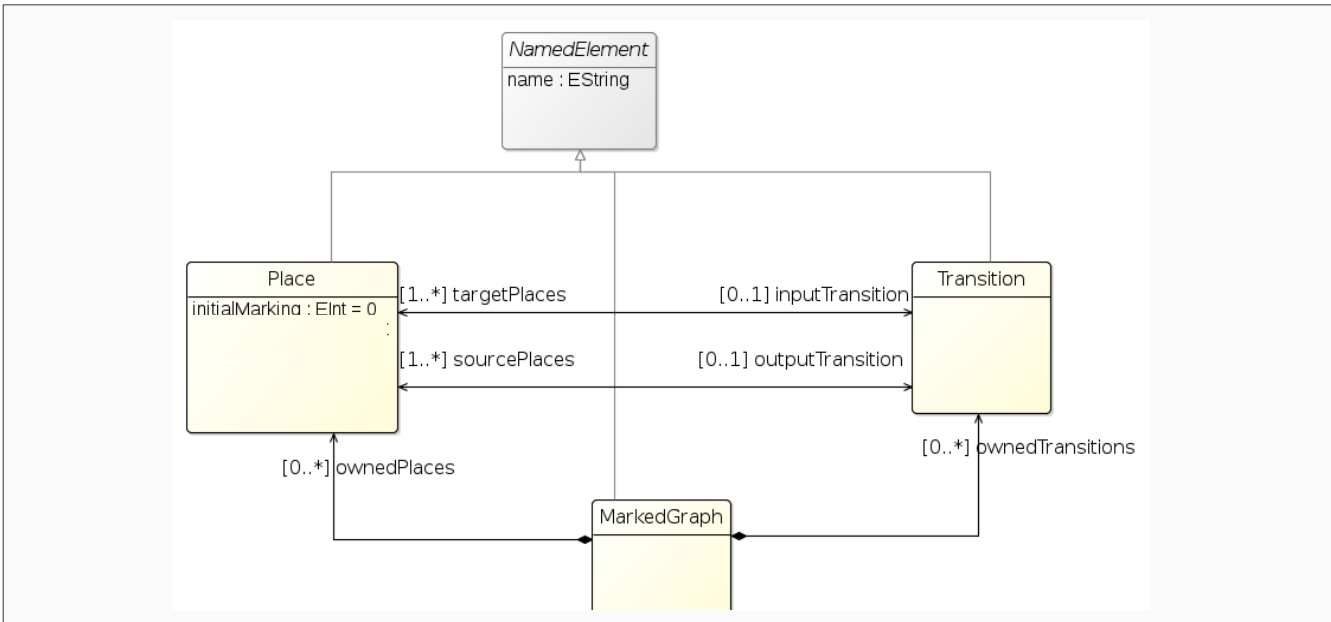
Language Workbench

Edit and debug your heterogeneous models

Design and compose your executable DSMLs

Running example: AS

Ecore+Sirius



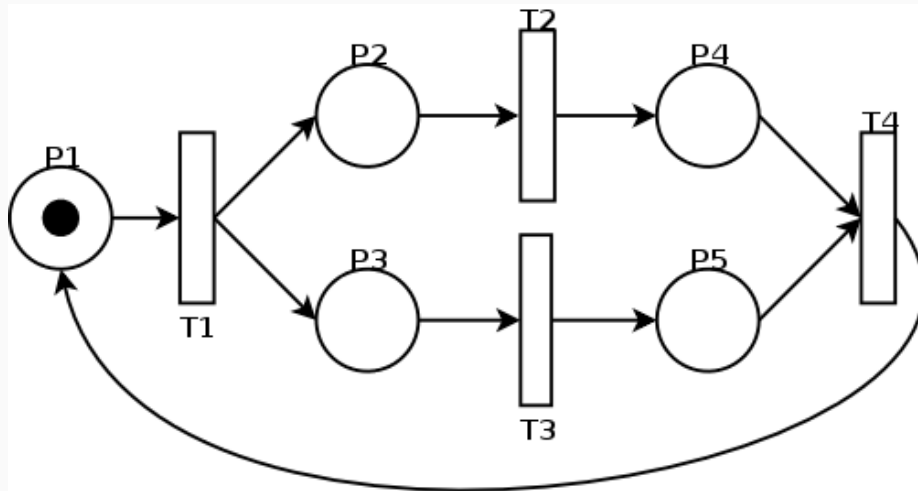
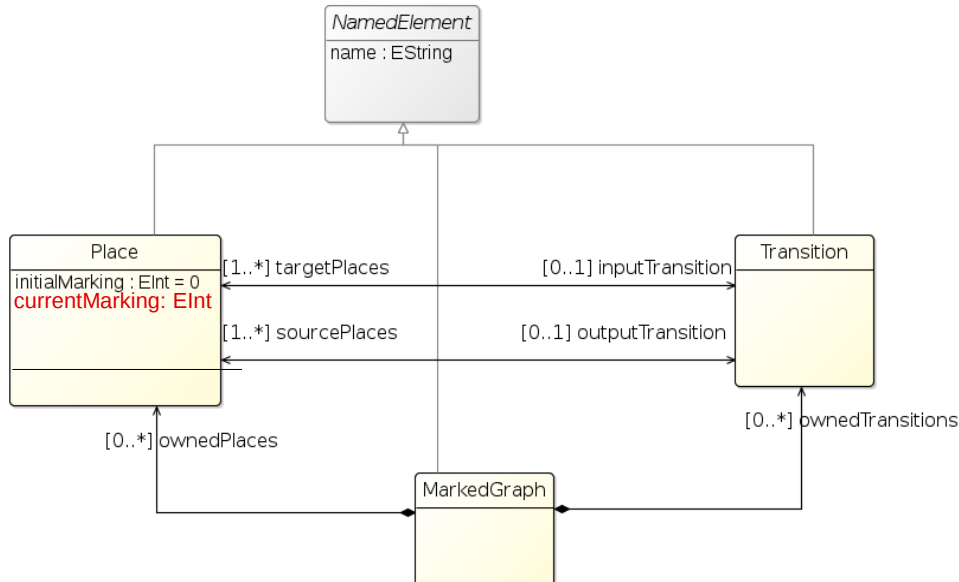
Running example: AS+DSA

Kermeta3

Domain Specific Action

(model state)

- The **current marking** represents the runtime state of this simple language

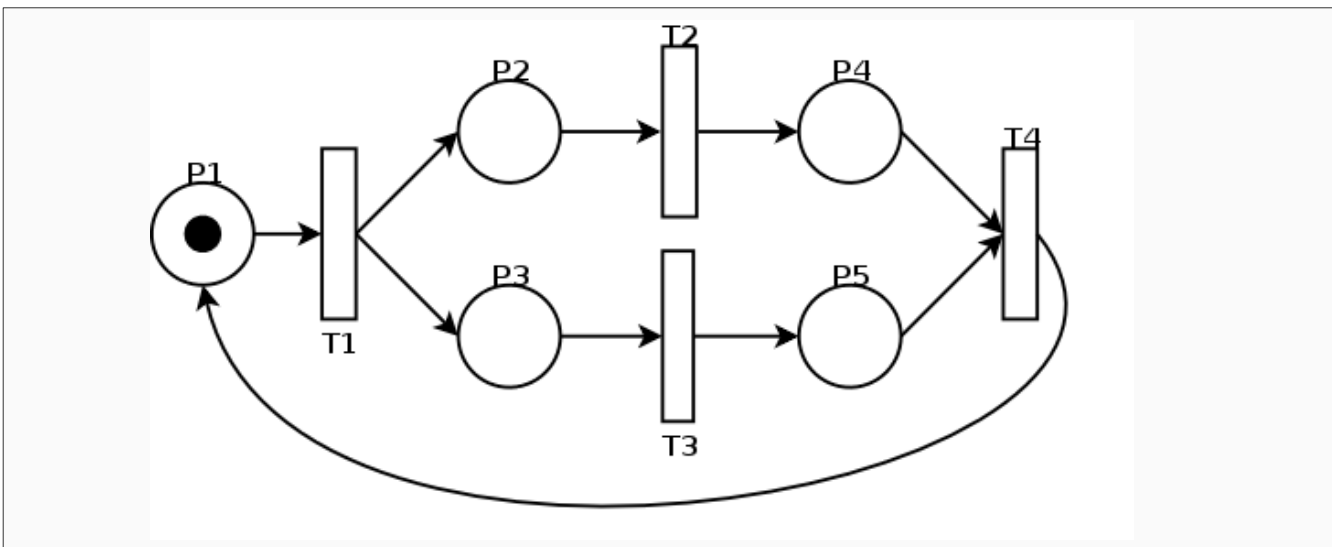
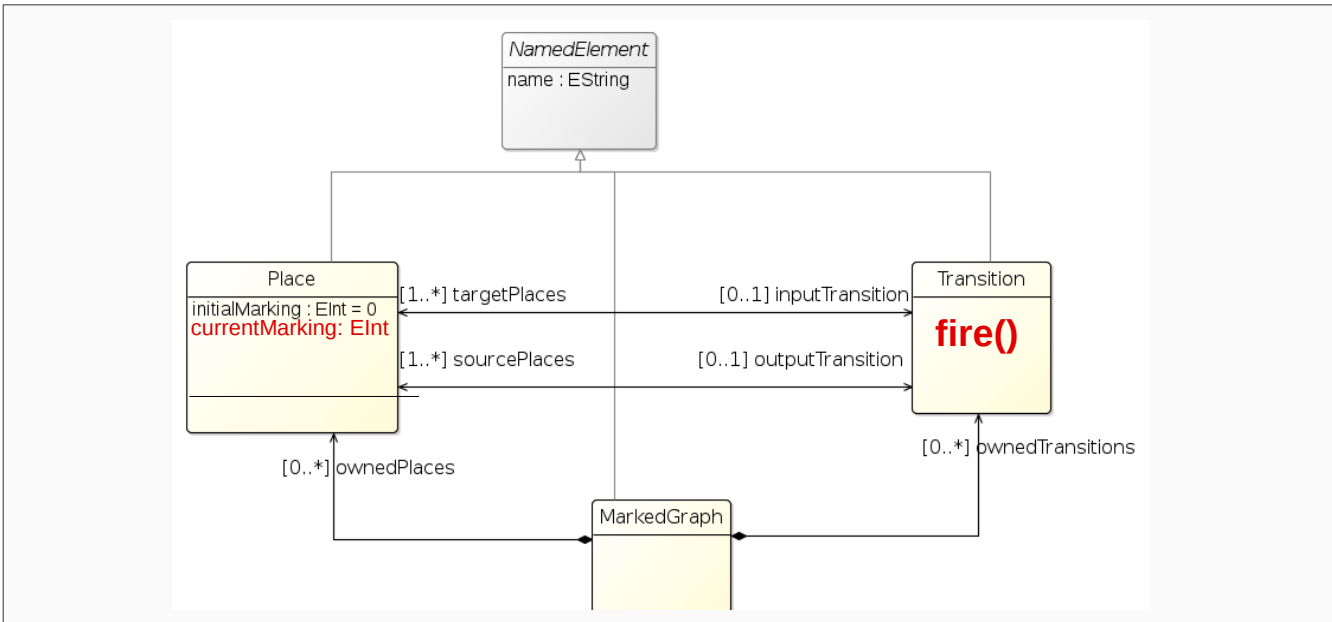


Running example: AS+DSA

Kermeta3

Domain Specific Action (execution functions)

```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```

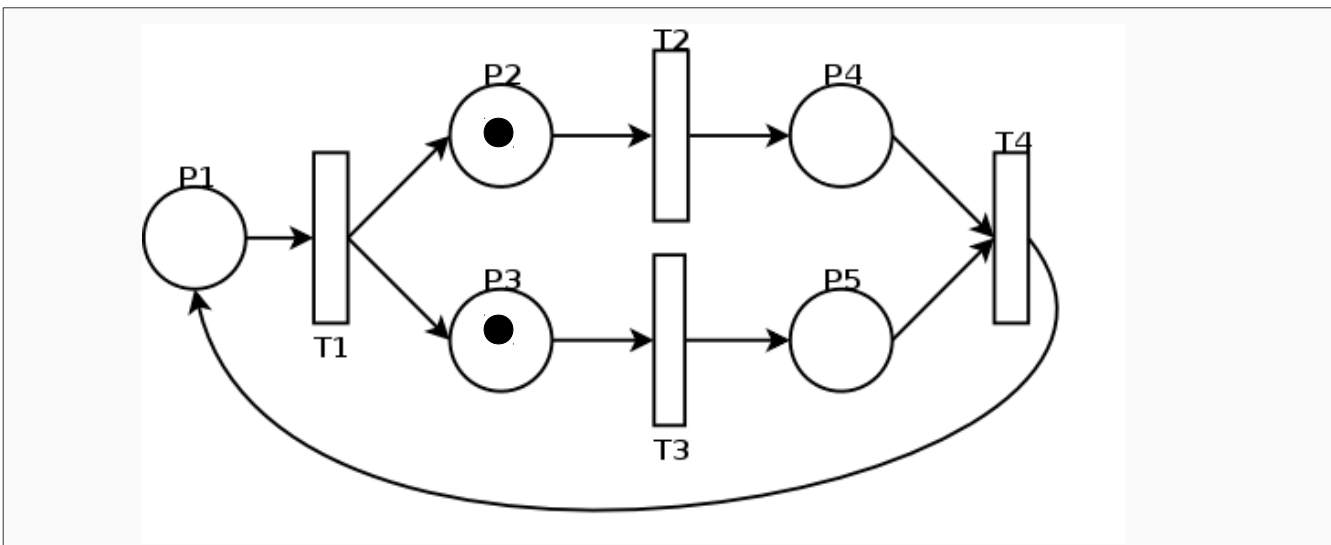
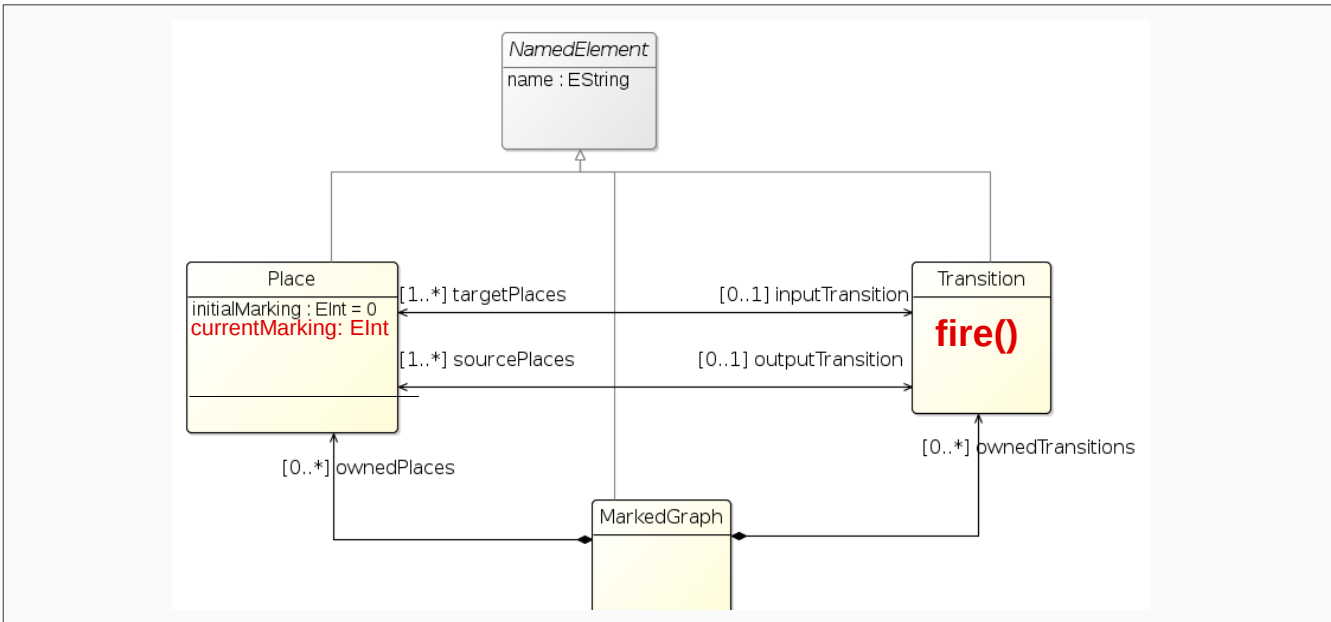


Running example: AS+DSA

Kermeta3

Domain Specific Action
(execution functions)

```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```

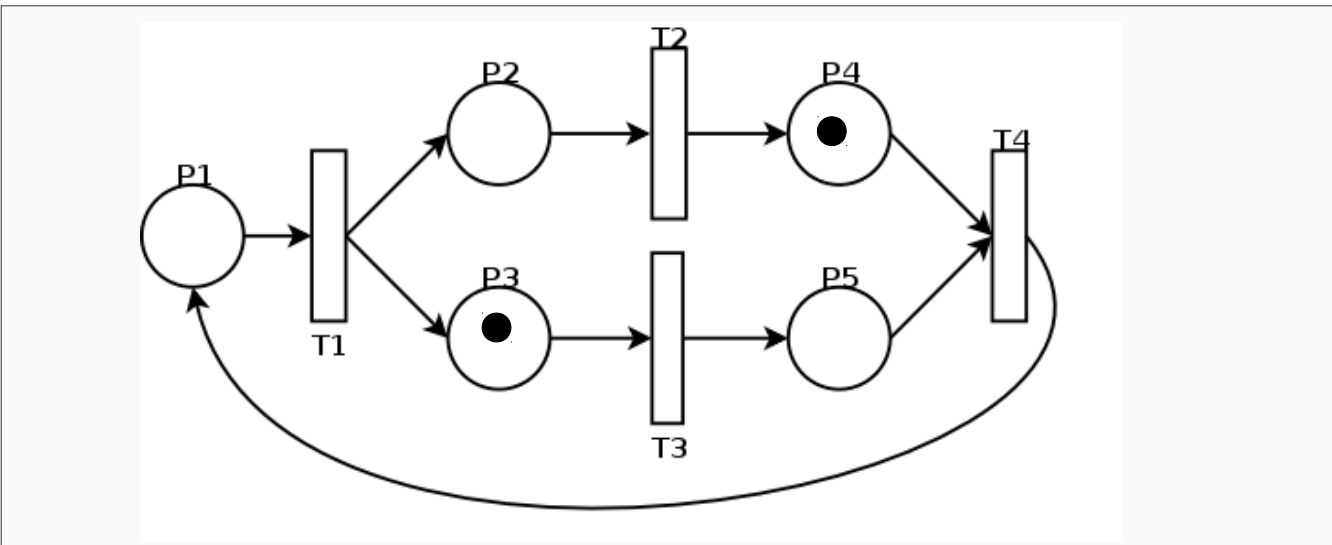
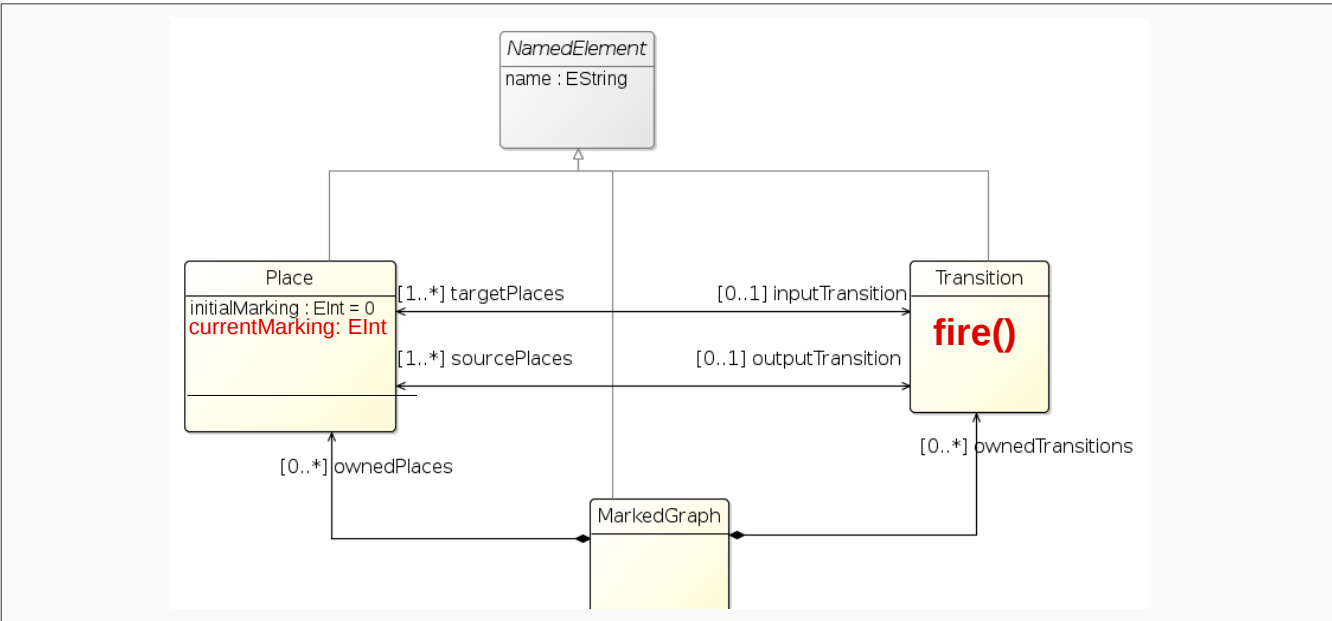


Running example: AS+DSA

Kermeta3

Domain Specific Action (execution functions)

```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```

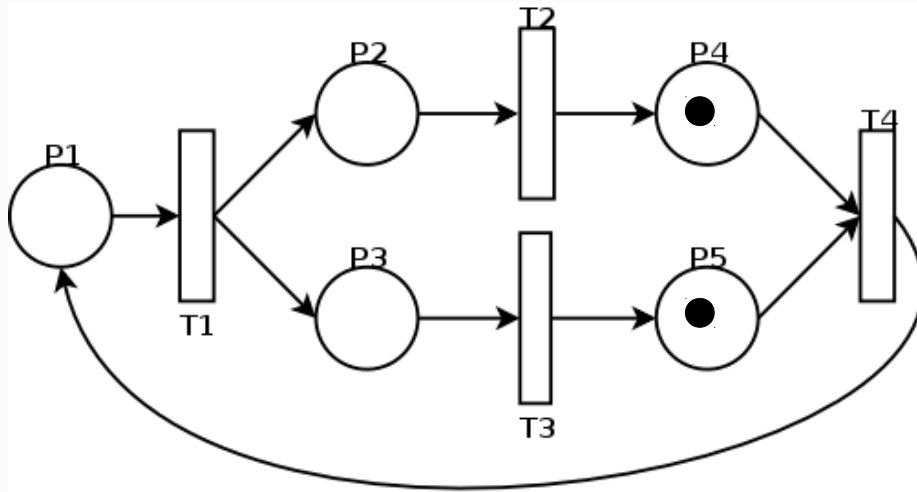
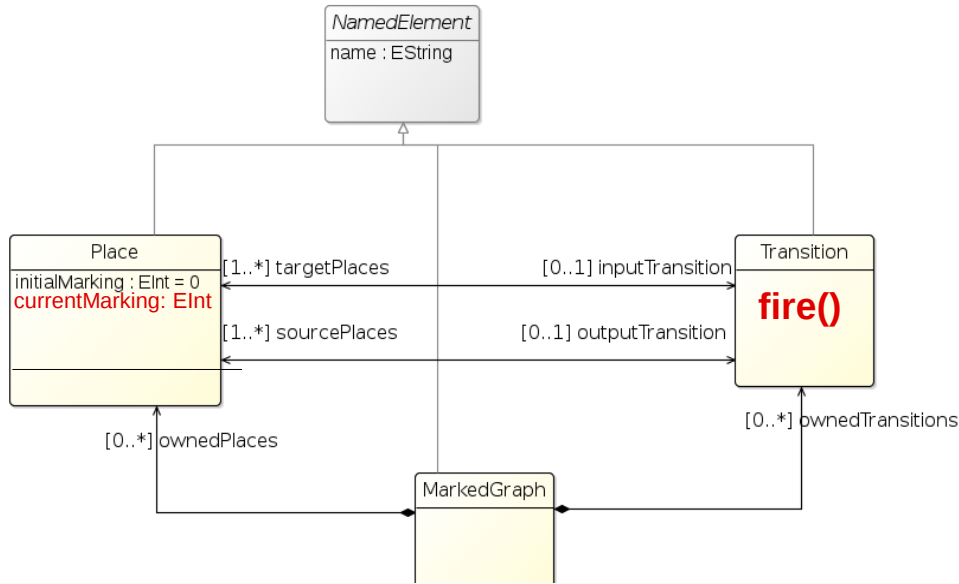


Running example: AS+DSA

Kermeta3

Domain Specific Action (execution functions)

```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```

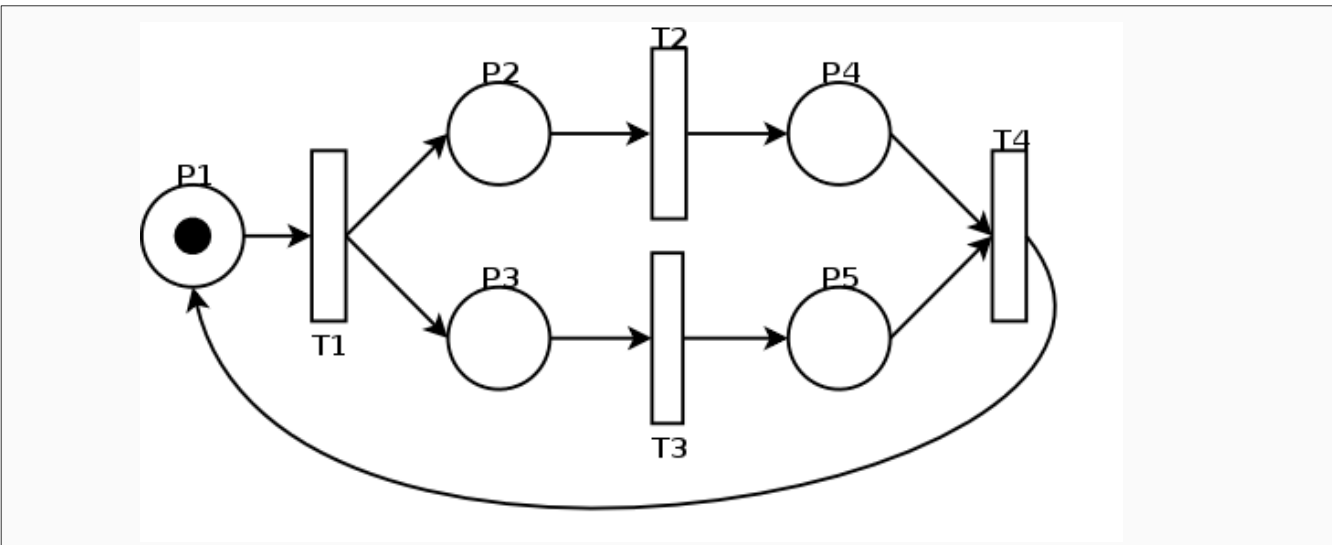
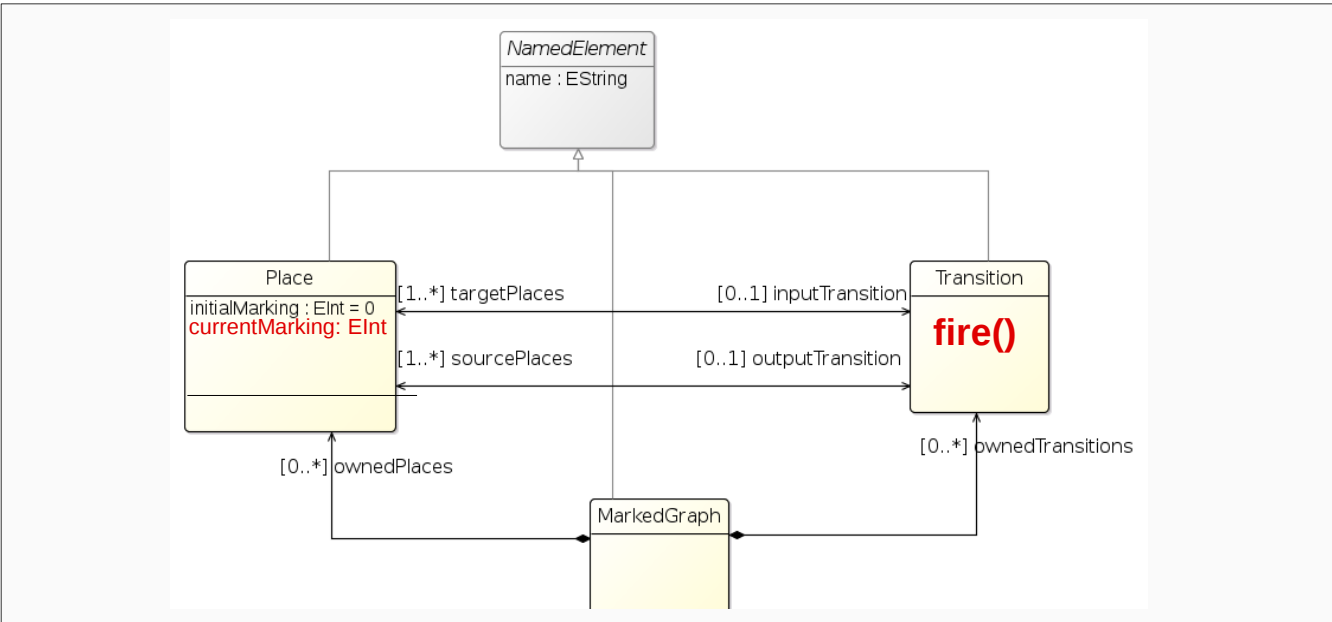


Running example: AS+DSA

Kermeta3

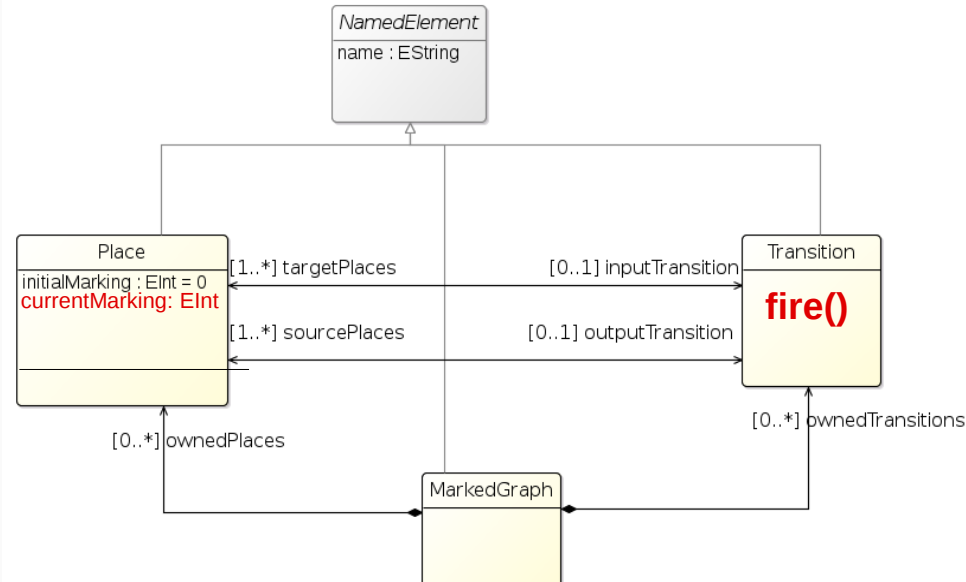
Domain Specific Action (execution functions)

```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```



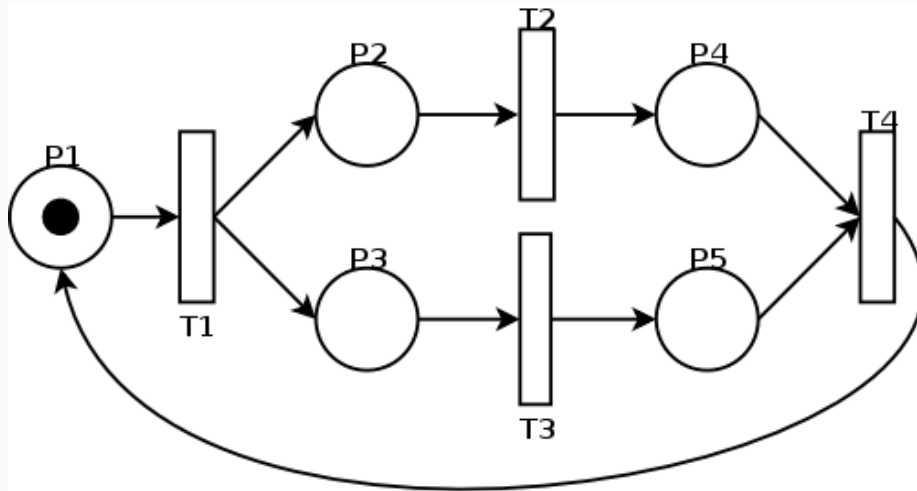
Running example: AS+DSA

Kermeta3



Domain Specific Action (execution functions)

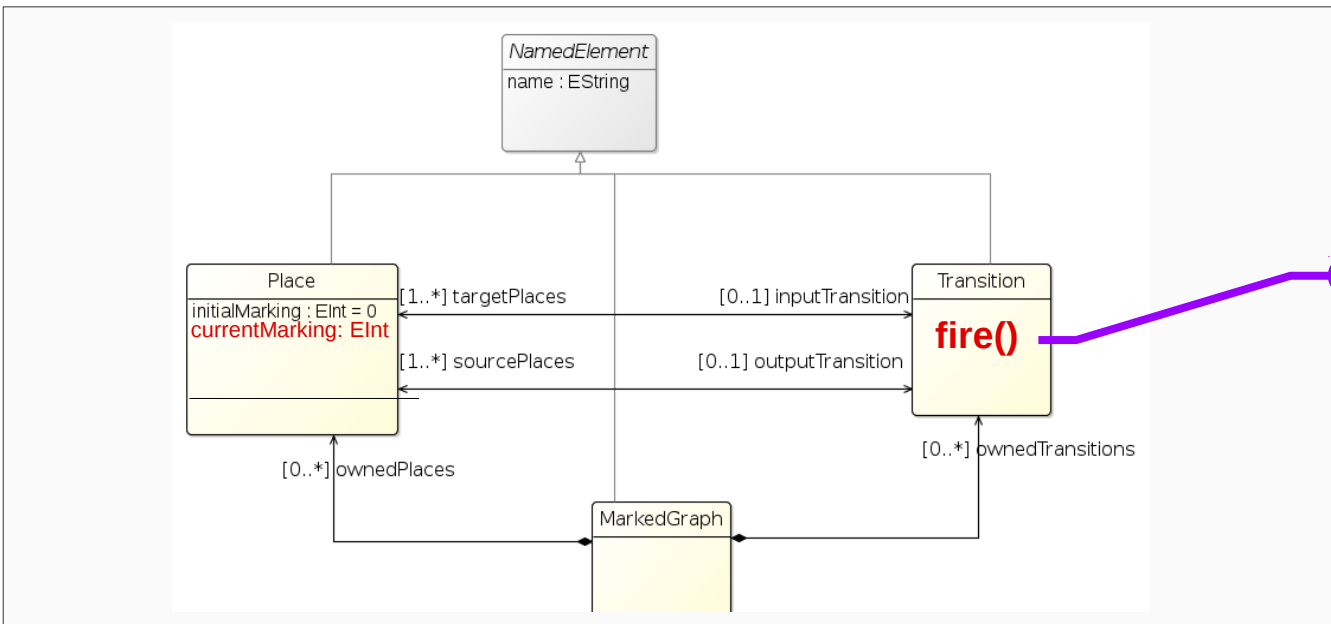
```
def fire(){
    _self.sourcePlaces.forEach [
        currentMarking --
    ]
    _self.targetPlaces.forEach [
        currentMarking ++
    ]
}
```



Nobody calls the *fire()* operation.
This is the model of concurrency and
causality that specifies **when** things
happen

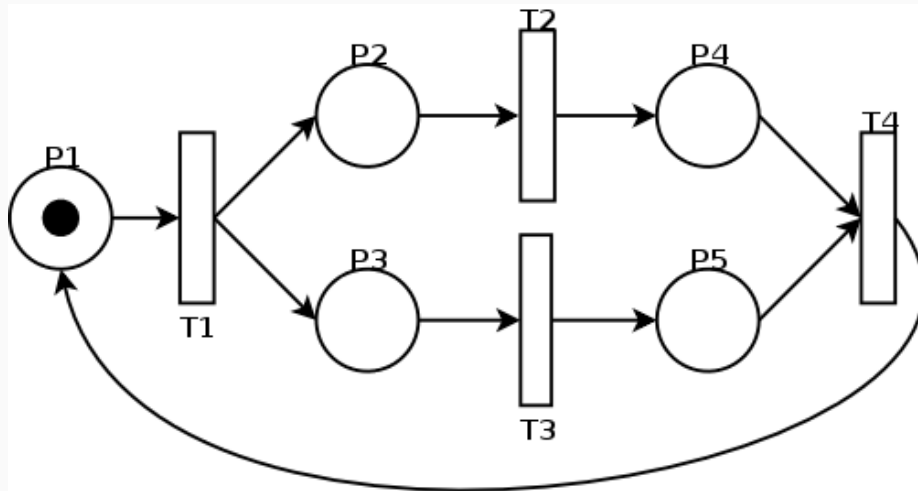
Running example: AS+DSA+DSE

ECL



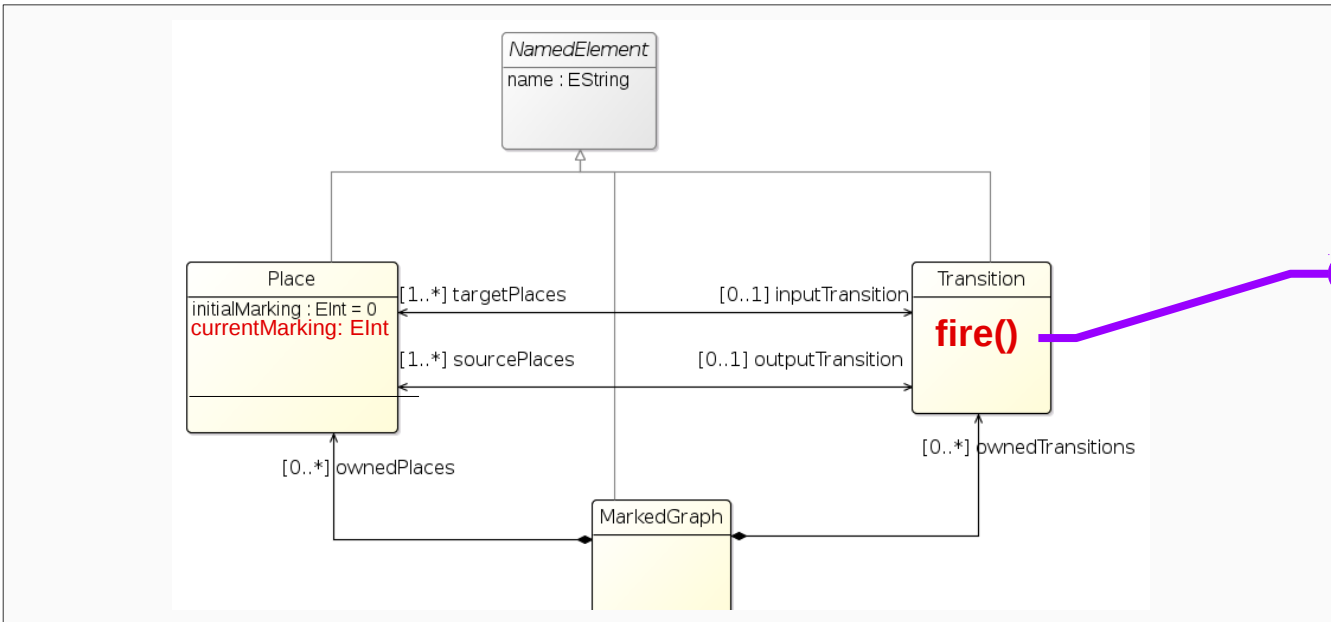
● fire(): DSE

Domain Specific Events
act as “handles” to the
DSA



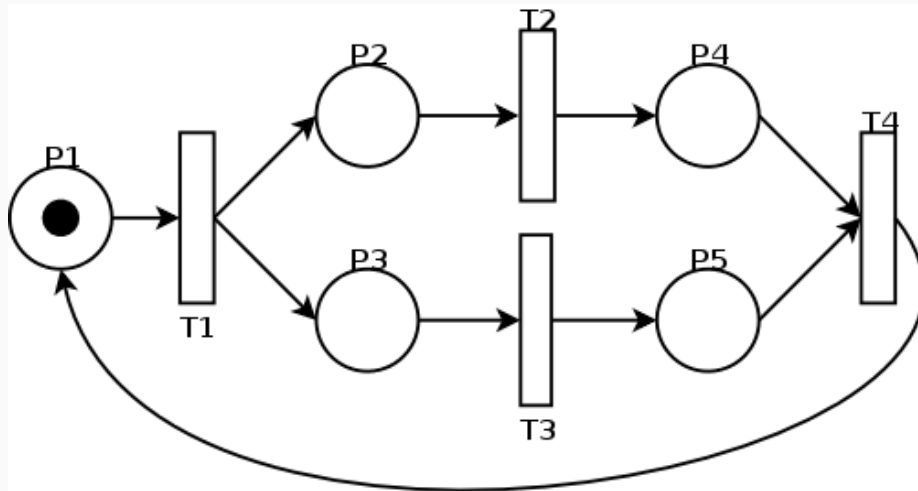
Running example: AS+DSA+DSE

MoCCML mapping (ex ECL)



● firelt: DSE

Domain Specific Events
act as “handles” to the
DSA



● fire_T1: firelt

● fire_T2: firelt

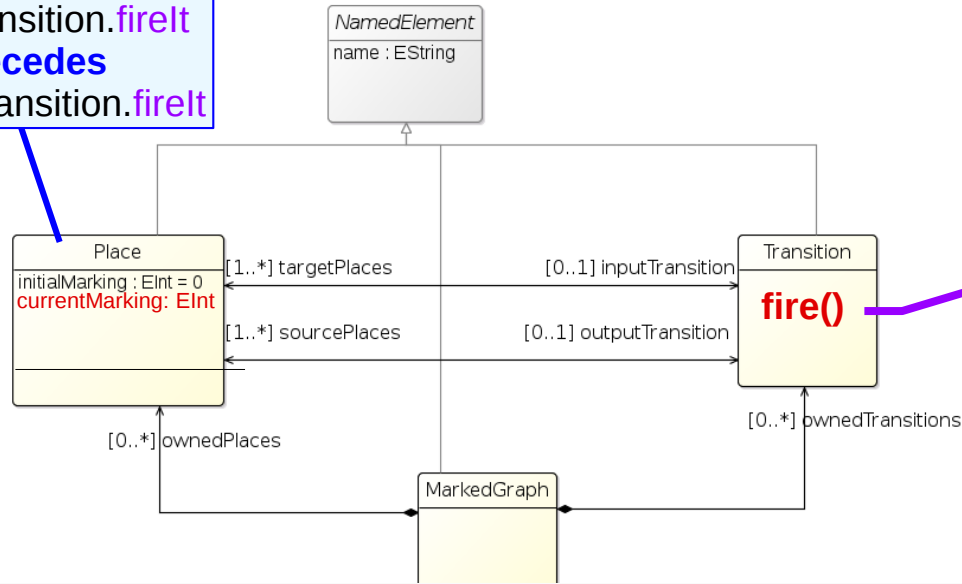
● fire_T3: firelt

● fire_T4: firelt

Running example: AS+DSA+DSE+MoCC

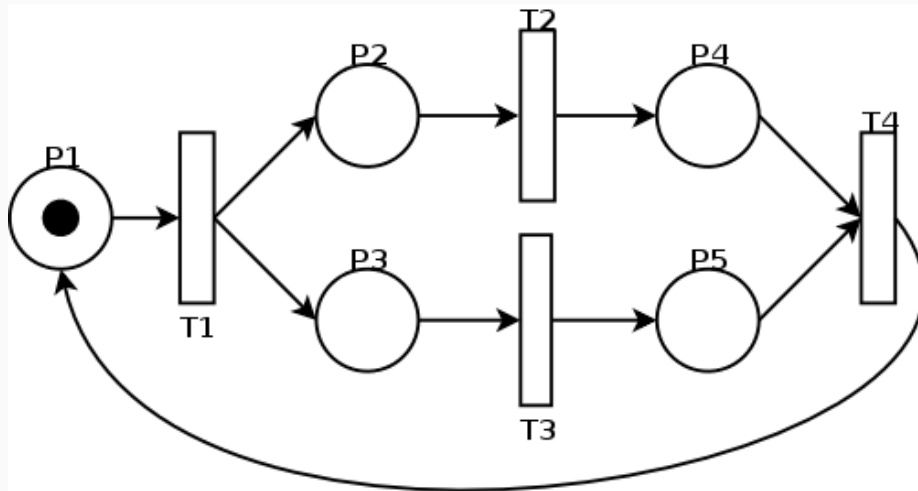
MoCCML

inputTransition.firelt
precedes
outputTransition.firelt



● firelt: DSE

The MoCC constrains the DSE and consequently defines the acceptable schedules of the actions



● fire_T1: firelt

● fire_T2: firelt

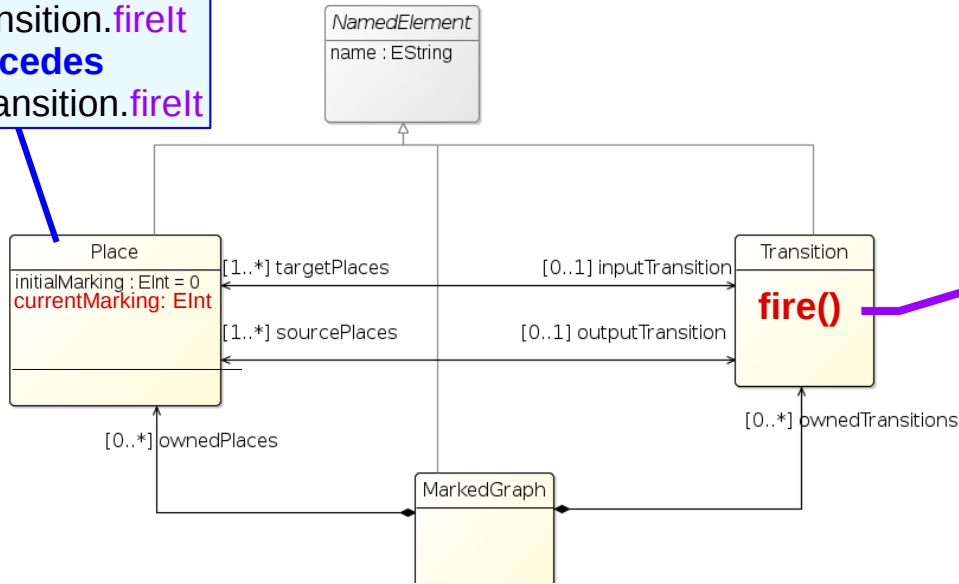
● fire_T3: firelt

● fire_T4: firelt

Running example: AS+**DSA**+**DSE**+MoCC

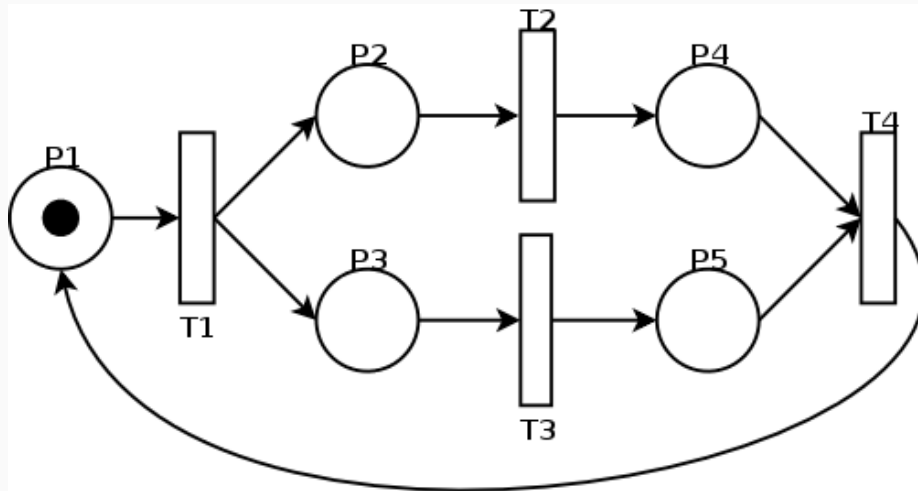
MoCCML

inputTransition.fireelt
precedes
outputTransition.fireelt



fireelt: DSE

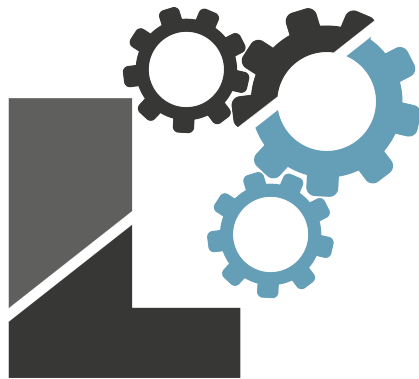
The MoCC constrains the DSE and consequently defines the acceptable schedules of the actions



- precedes ● fire_T1: firelt
- precedes ● fire_T2: firelt
- precedes ● fire_T3: firelt
- precedes ● fire_T4: firelt

The GEMOC Studio

Ecore+Sirius
Kermeta3
ECL
MoCCML



Language Workbench

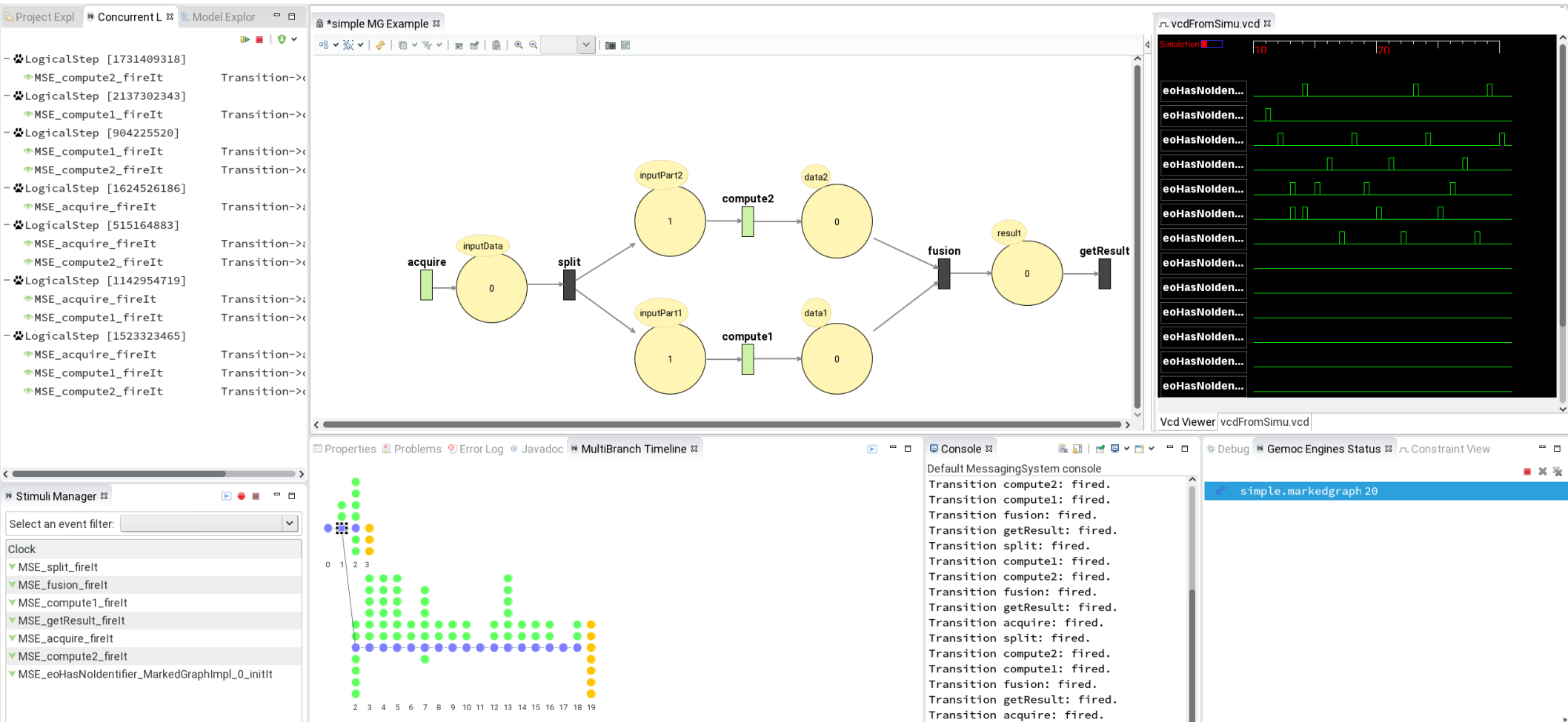


Automatic generation



Modeling Workbench

The GEMOC Studio



The screenshot displays the GEMOC Studio interface with several key components:

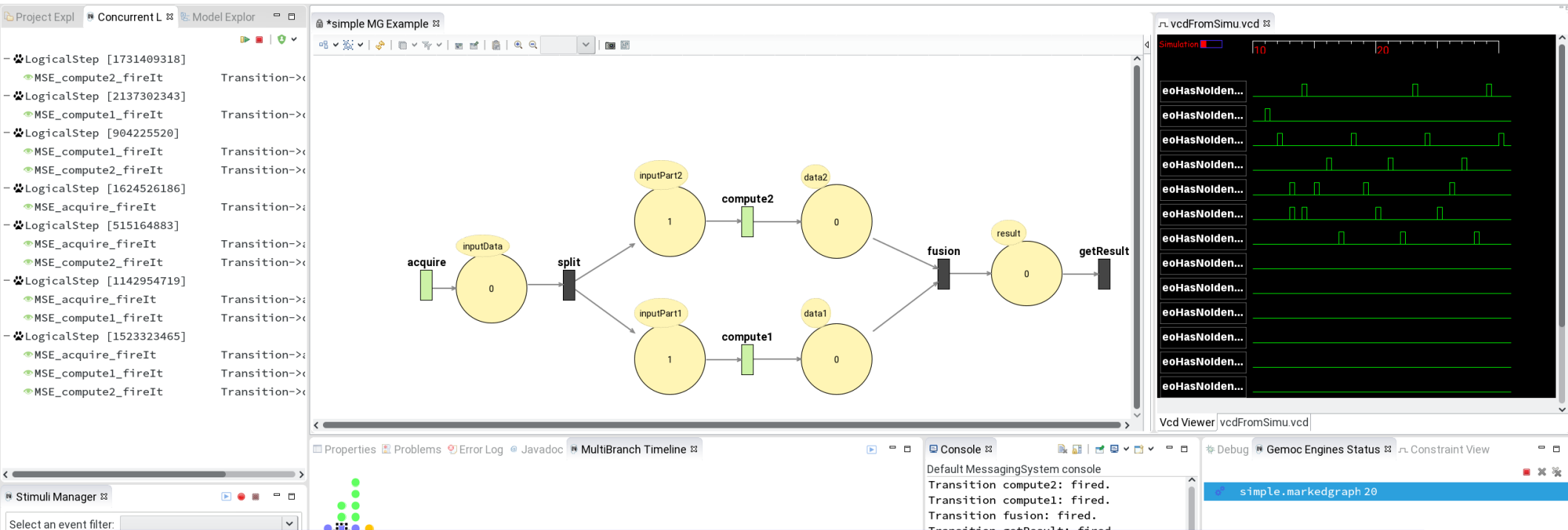
- Project Explorer:** Shows a list of LogicalSteps and their associated transitions (e.g., MSE_compute2_fireIt, MSE_compute1_fireIt).
- Model Explorer:** Displays a Marked Graph (MG) model with nodes (inputData, inputPart1, inputPart2, data1, data2, result) and transitions (acquire, split, compute1, compute2, fusion, getResult).
- Simulation View:** A Vcd Viewer showing a simulation timeline with multiple instances of the event 'eoHasNoiden...'.
- Console:** A log of events such as 'Transition compute2: fired.', 'Transition compute1: fired.', 'Transition fusion: fired.', and 'Transition getResult: fired.'.
- Stimuli Manager:** A panel for selecting event filters and viewing a clock.
- MultiBranch Timeline:** A visualization of the simulation's state over time, showing the firing of various transitions.

Language Workbench

Automatic generation

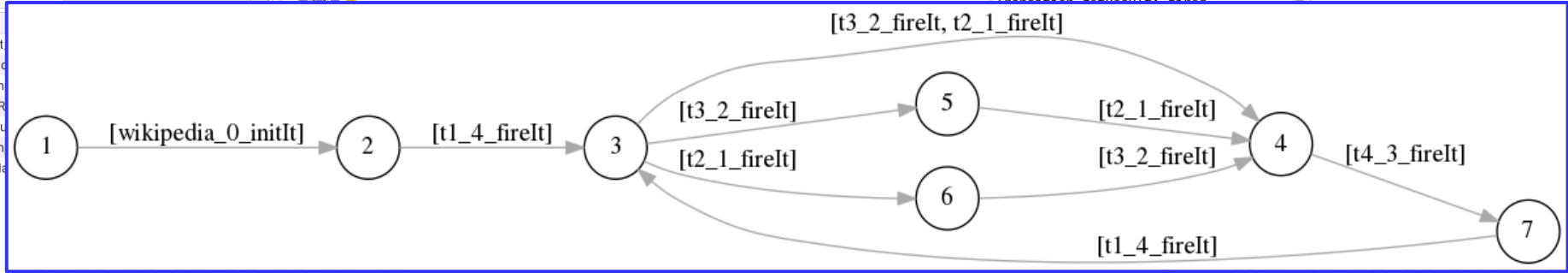
Modeling Workbench

The GEMOC Studio



The screenshot displays the GEMOC Studio environment. The central window shows a Marked Graph (MG) model with nodes and transitions. The nodes include 'inputData', 'split', 'inputPart1', 'compute1', 'data1', 'inputPart2', 'compute2', 'data2', 'fusion', 'result', and 'getResult'. The transitions are labeled 'acquire', 'compute1', 'compute2', 'fusion', and 'getResult'. The left pane shows a list of LogicalSteps and their corresponding transitions. The right pane shows a Vcd Viewer with a simulation waveform for 'eoHasNoiden...'. The bottom pane shows a Console window with the following output:

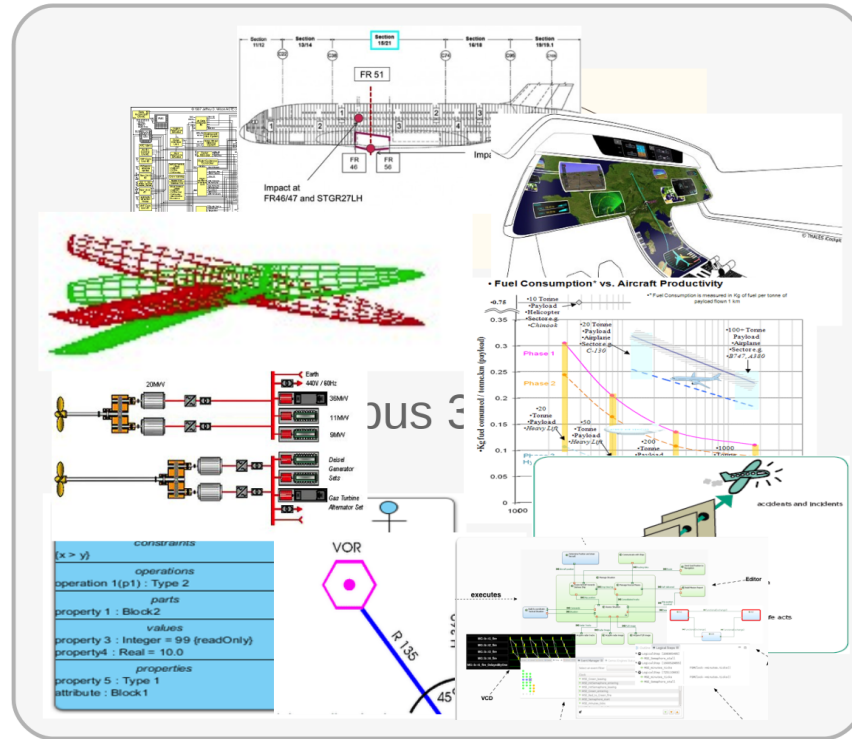
```
Default MessagingSystem console
Transition compute2: fired.
Transition compute1: fired.
Transition fusion: fired.
Transition getResult: fired.
```



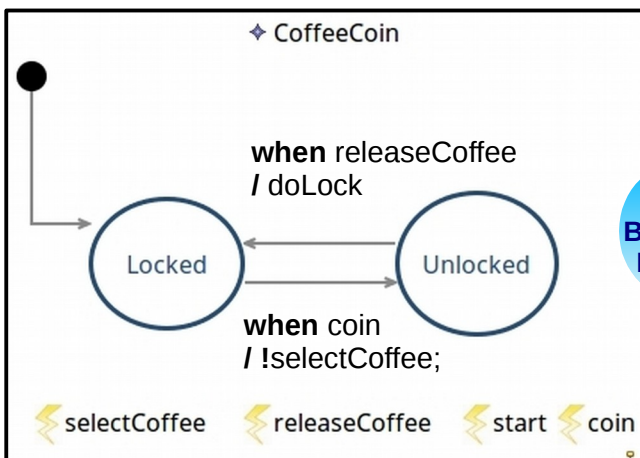
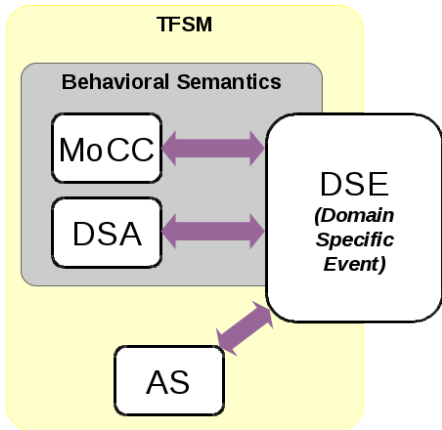
Language Workbench

Modeling Workbench

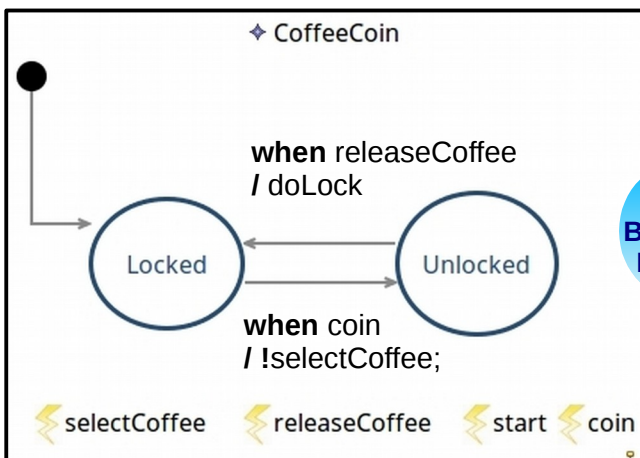
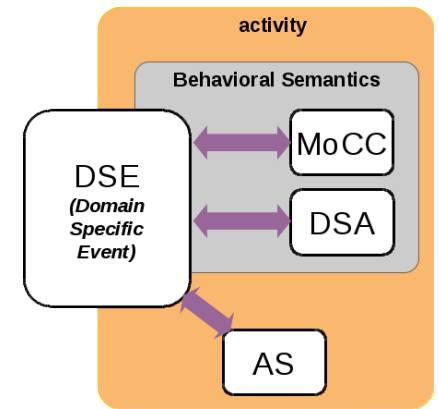
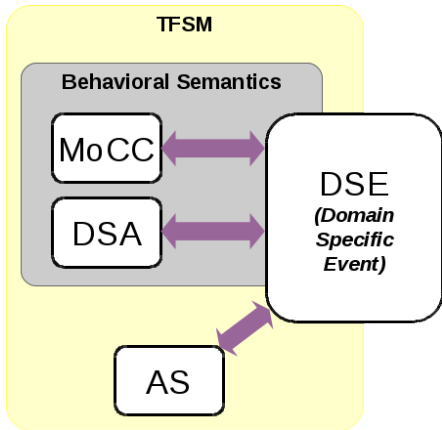
Model Based System Engineering



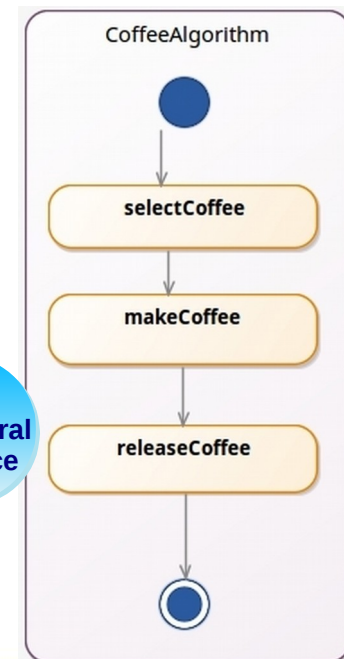
Model Based System Engineering specifies in a model the correspondences between models from the different concerns, all along the product life cycle.



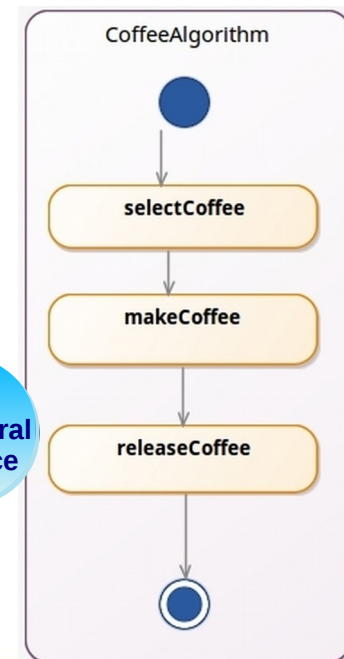
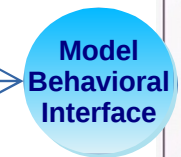
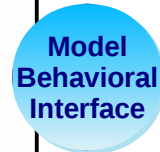
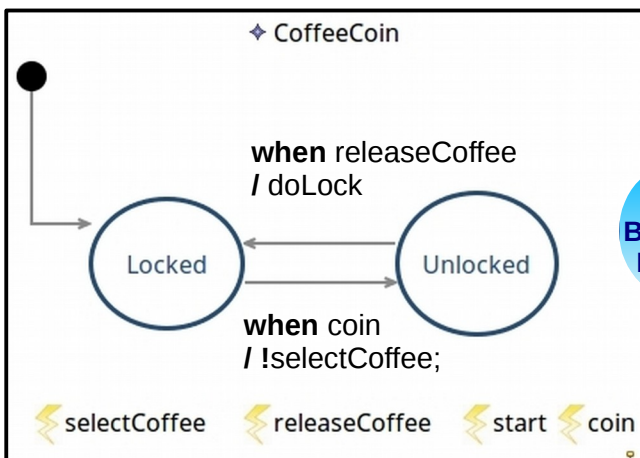
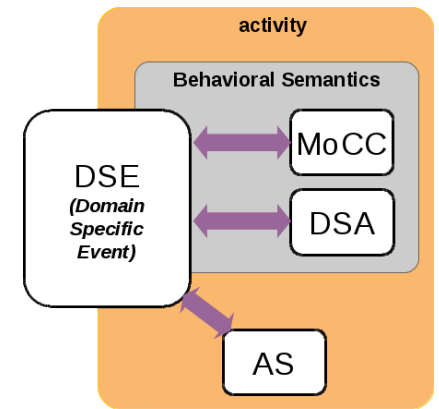
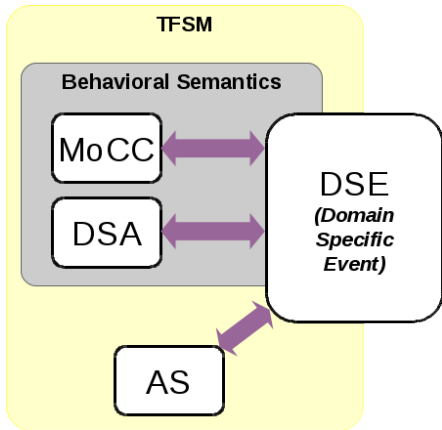
Model Behavioral Interface

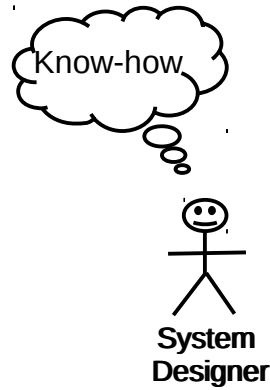
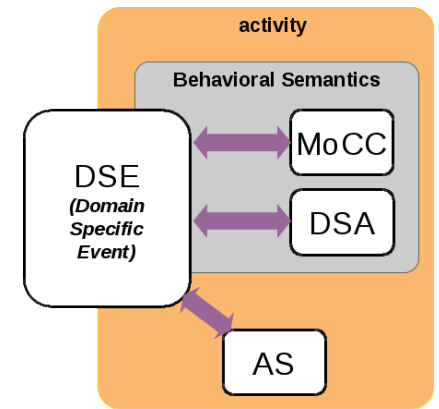
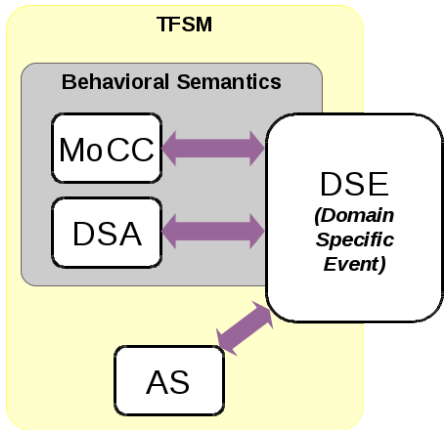


Model Behavioral Interface

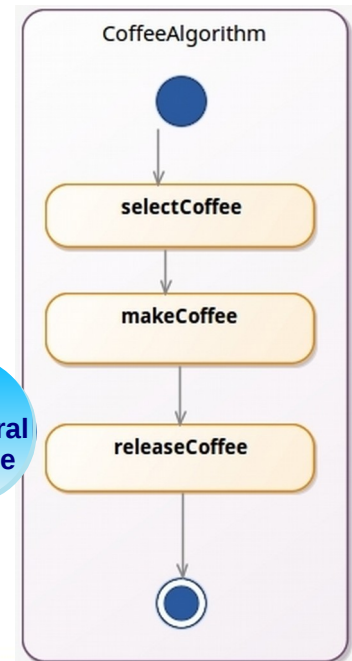
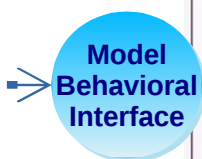
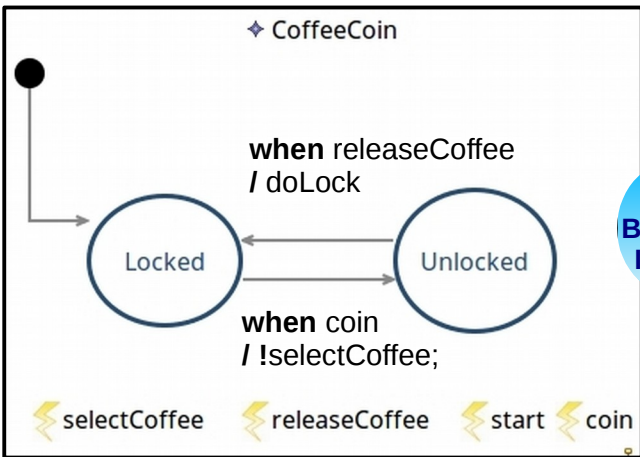
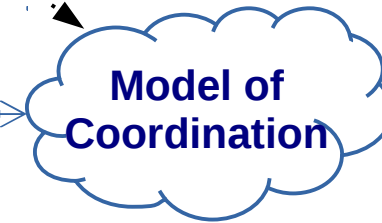


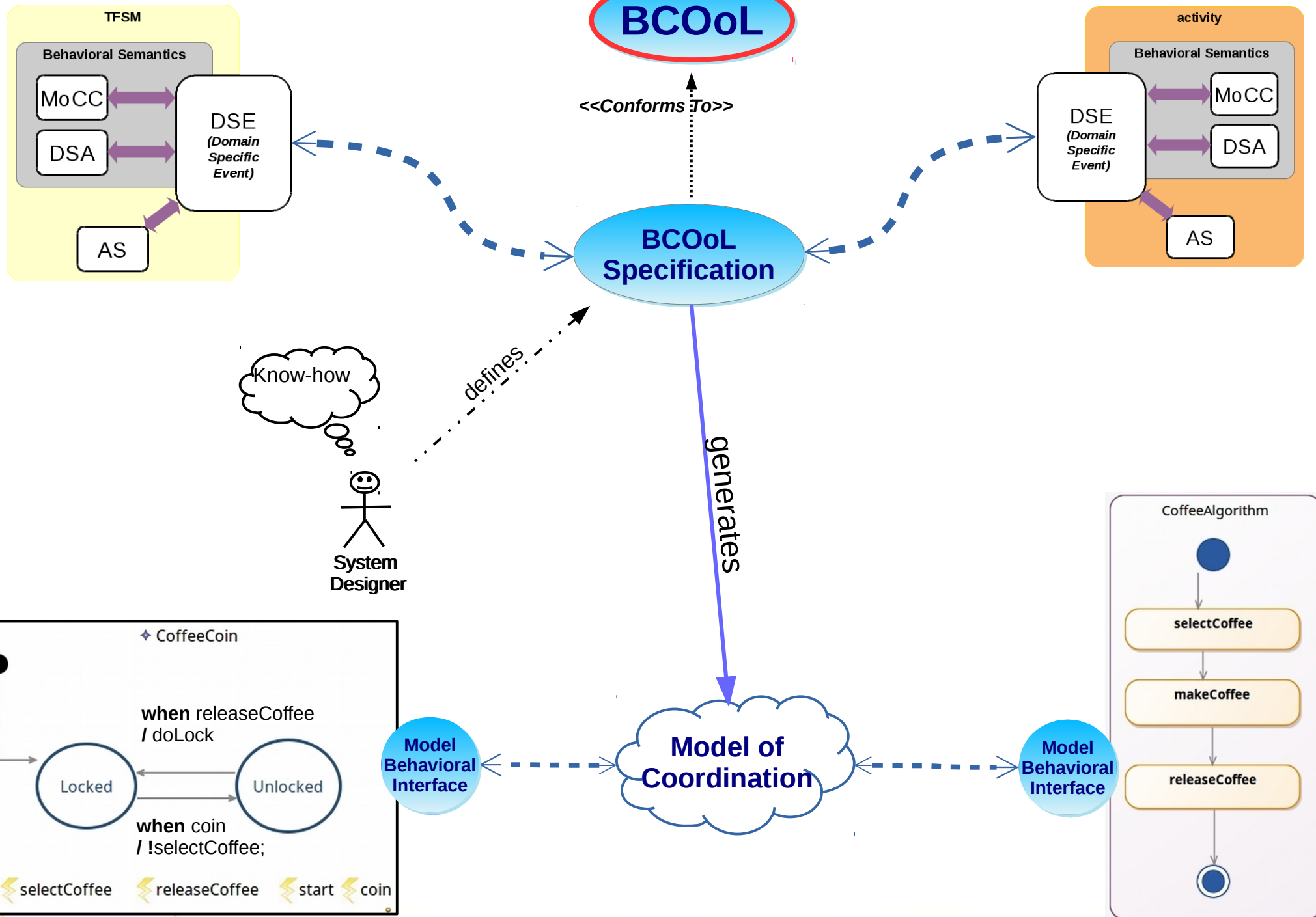
Model Behavioral Interface

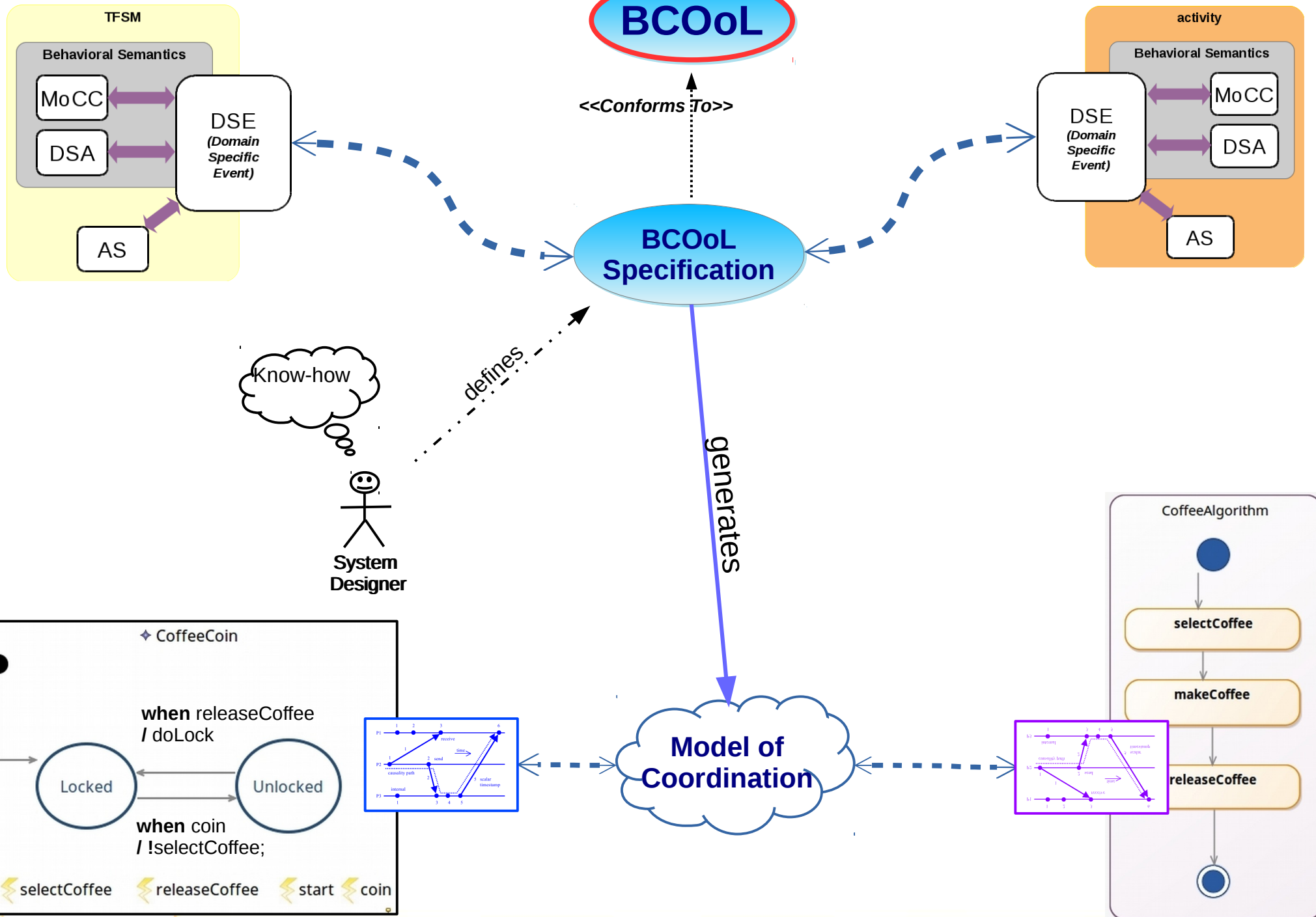


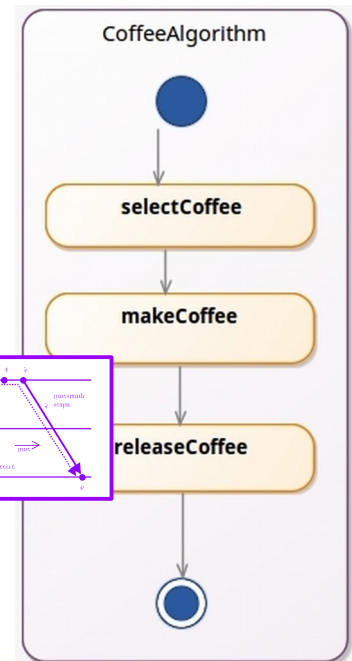
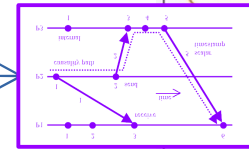
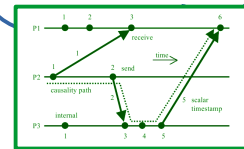
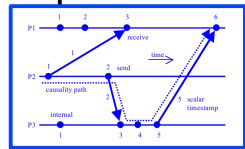
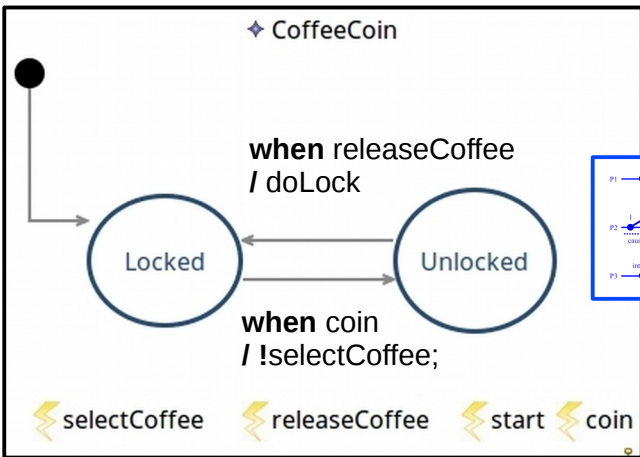
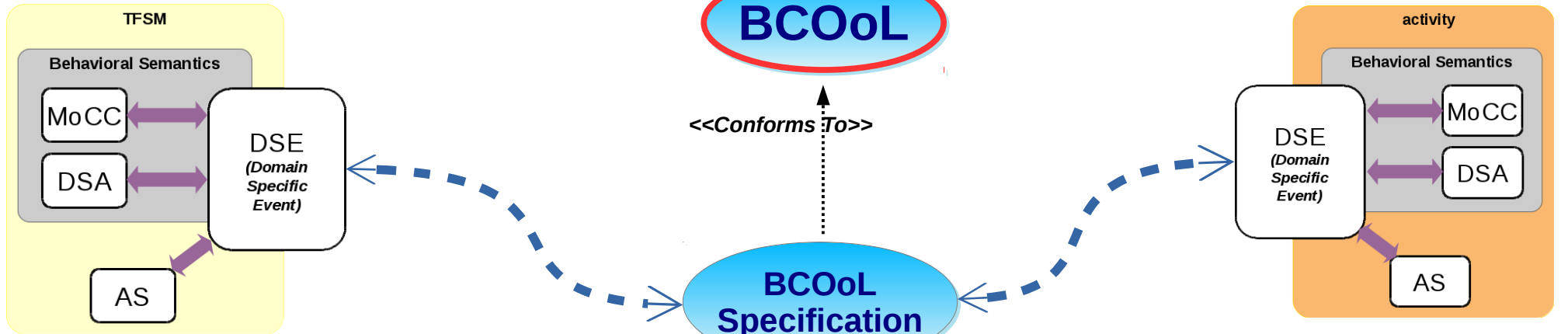


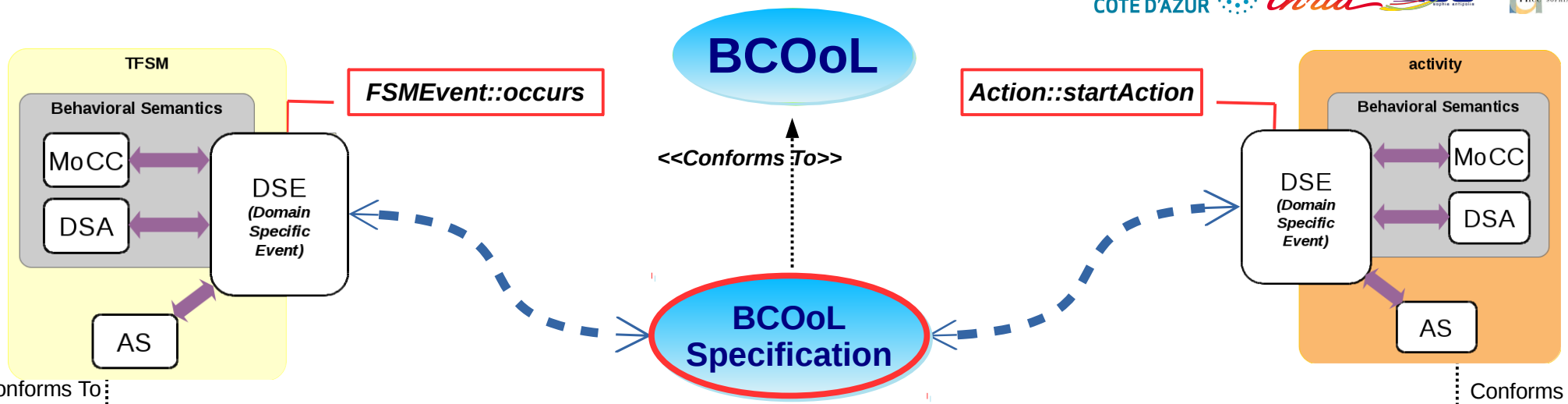
... defines ...





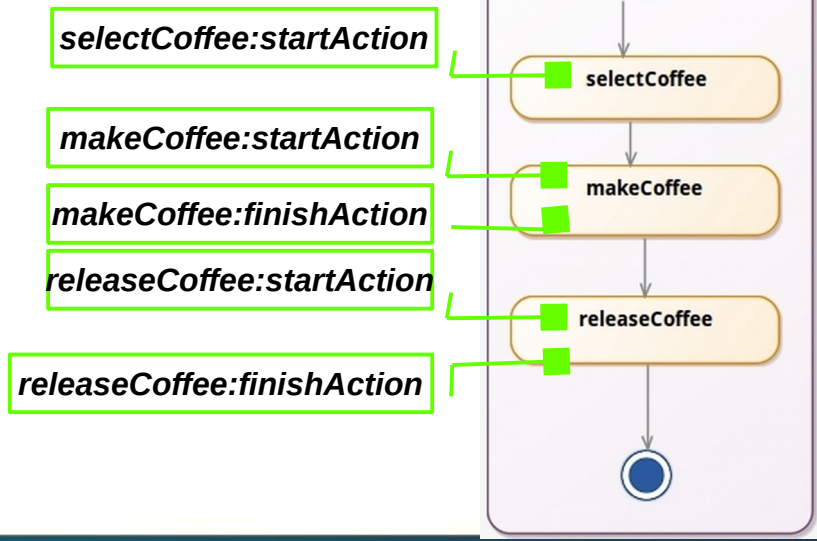
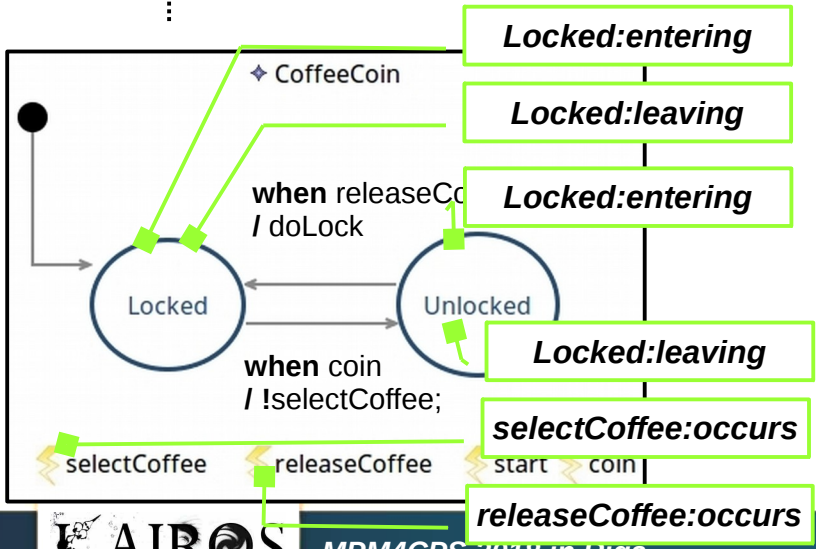




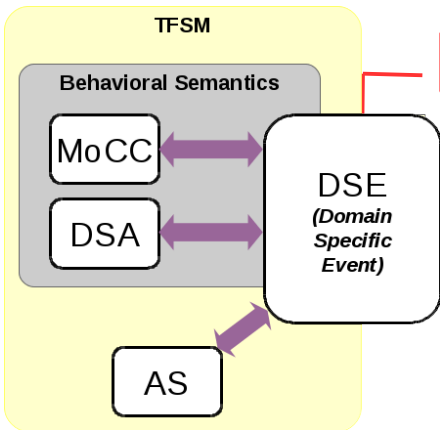


Conforms To

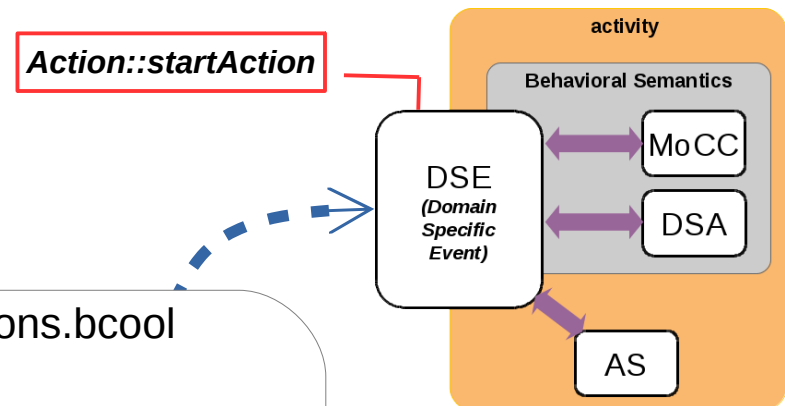
Conforms To



BCOoL



FSMEvent::occurs



Action::startAction

<<Conforms To>>

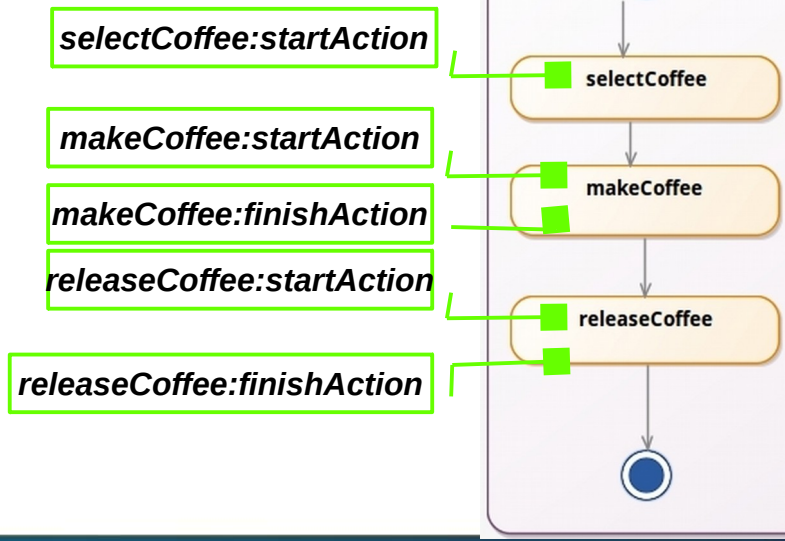
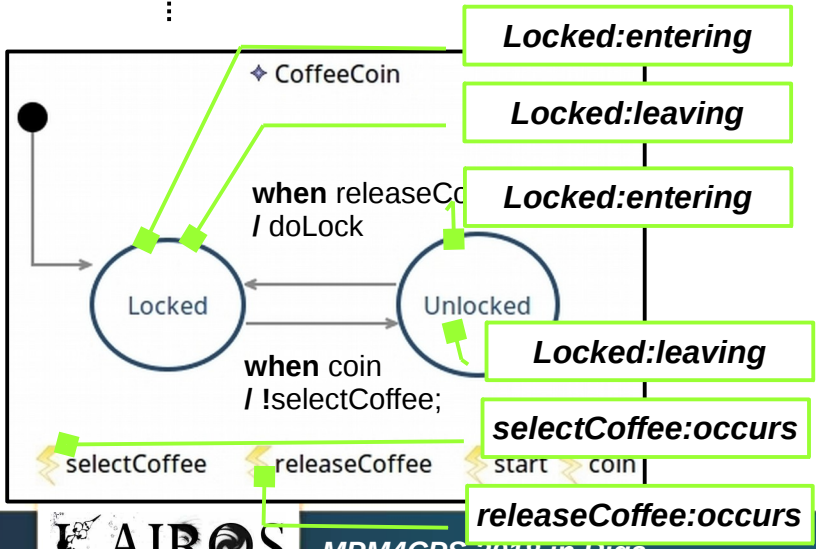
```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

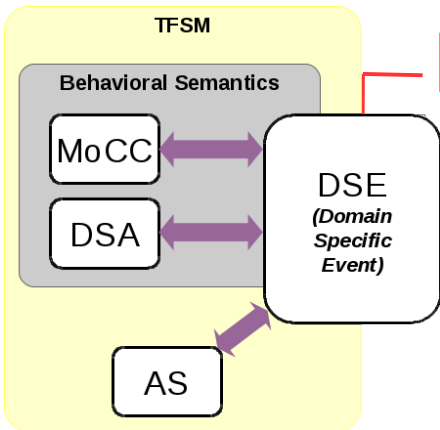
Operator FSMEventsandActions
  (FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

Conforms To

Conforms To

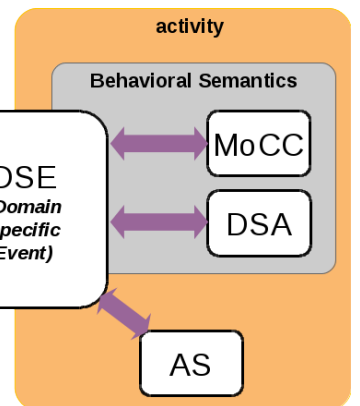


BCOoL



FSMEvent::occurs

Action::startAction



<<Conforms To>>

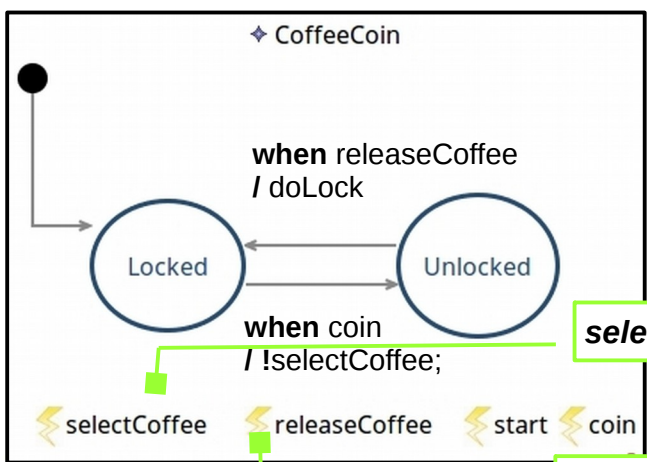
```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

Operator FSMEventsandActions
  (FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

Conforms To

Conforms To



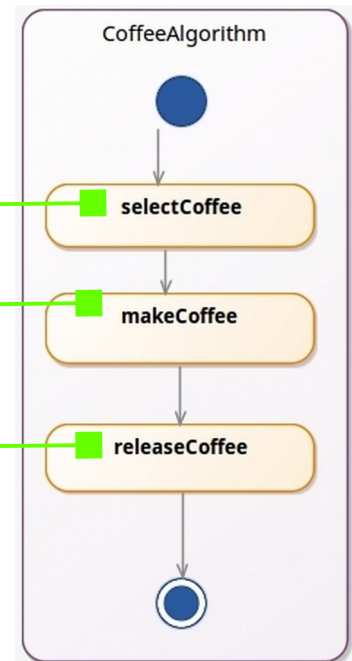
selectCoffee:occurs

releaseCoffee:occurs

selectCoffee:startAction

makeCoffee:startAction

releaseCoffee:startAction

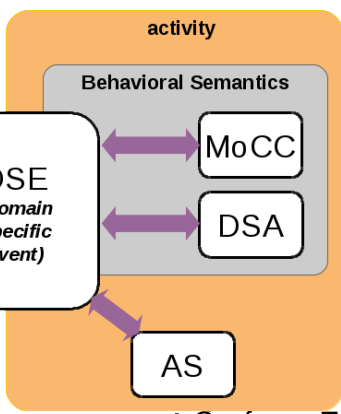
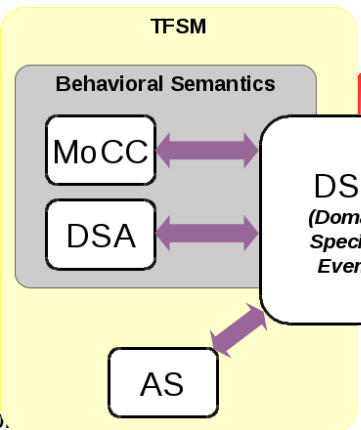


BCoOL

<<Conforms To>>

FSMEvent::occurs

Action::startAction



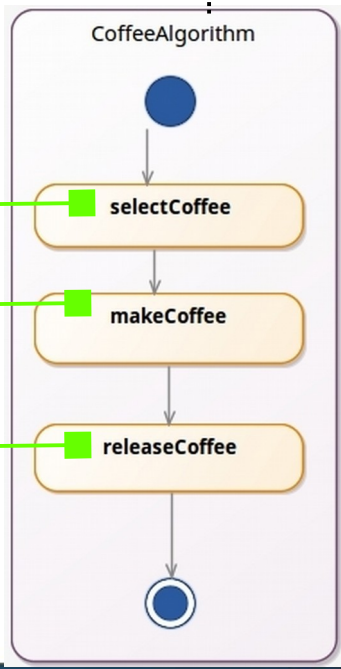
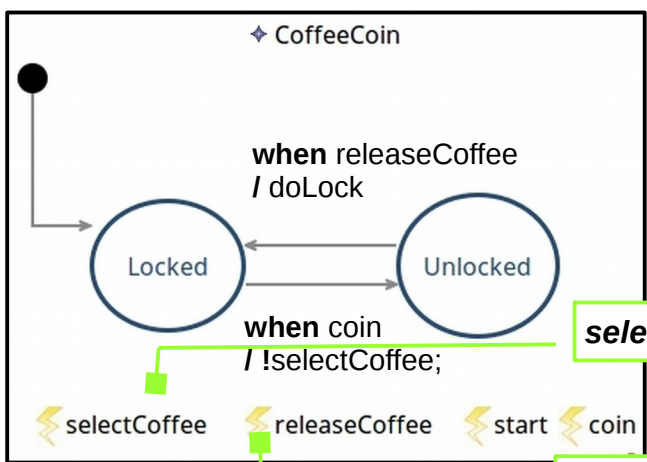
```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

Operator FSMEventsandActions
(FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

Confo.

Conforms To
Conforms To



selectCoffee:occurs

releaseCoffee:occurs

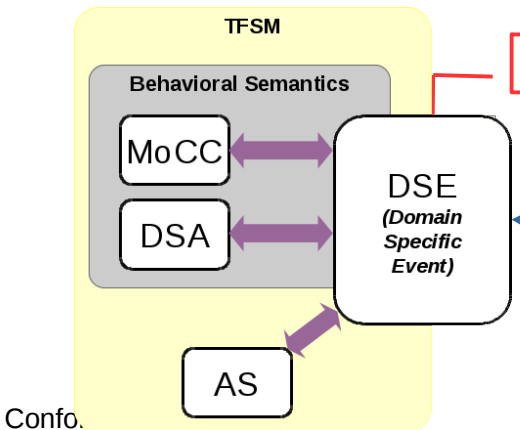
selectCoffee:startAction

makeCoffee:startAction

releaseCoffee:startAction

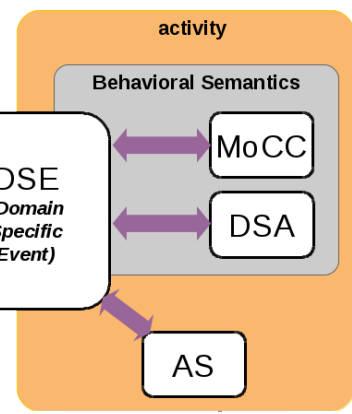
BCoOL

<<Conforms To>>



FSMEvent::occurs

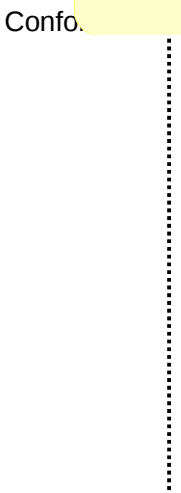
Action::startAction



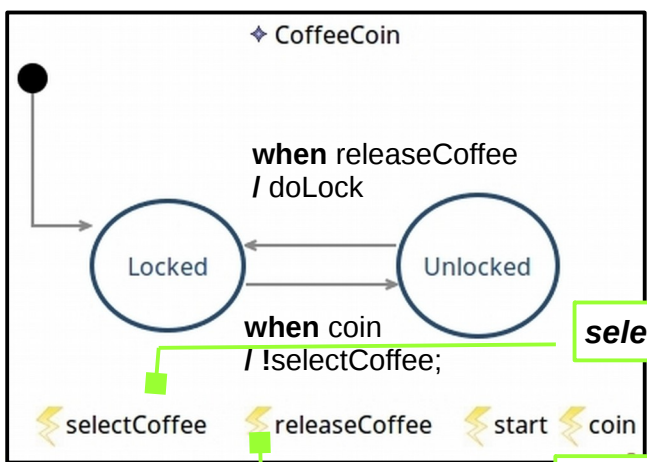
```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

Operator FSMEventsandActions
(FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

Confo. 

Conforms To 

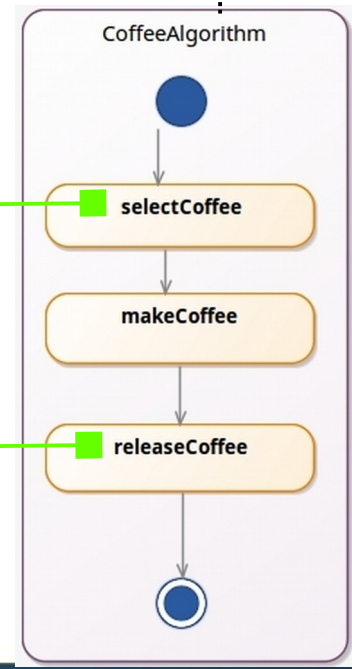


selectCoffee:startAction

releaseCoffee:startAction

selectCoffee:occurs

releaseCoffee:occurs

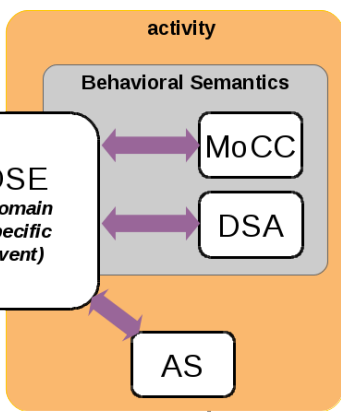
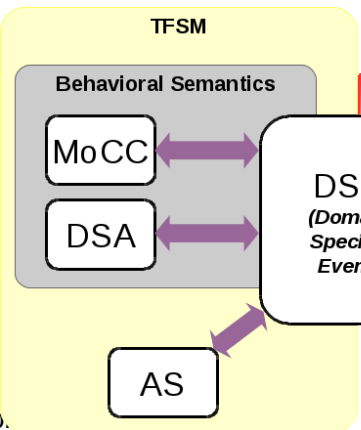


BCoOL

<<Conforms To>>

FSMEvent::occurs

Action::startAction

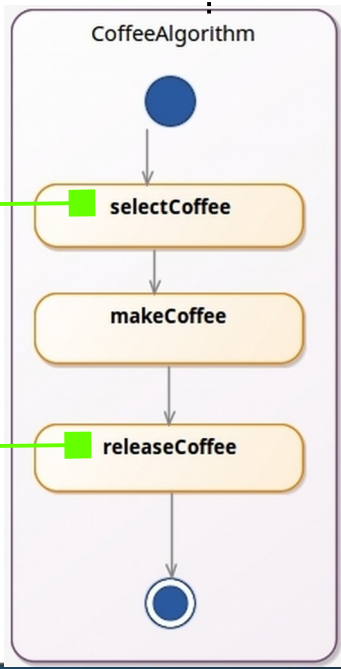
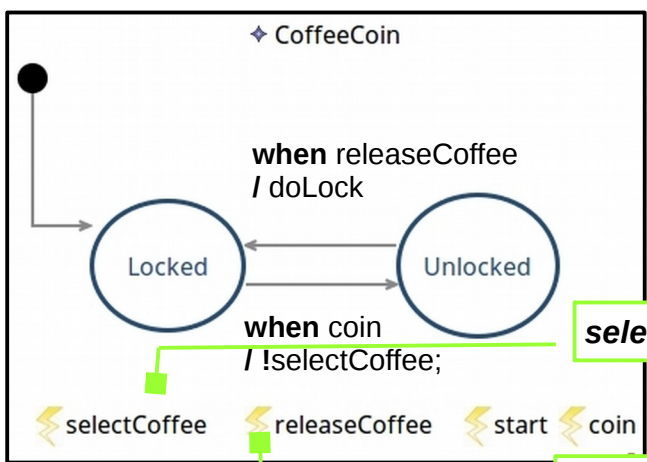


```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

Operator FSMEventsandActions
(FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

Ontology ??



selectCoffee:occurs

selectCoffee:startAction

releaseCoffee:startAction

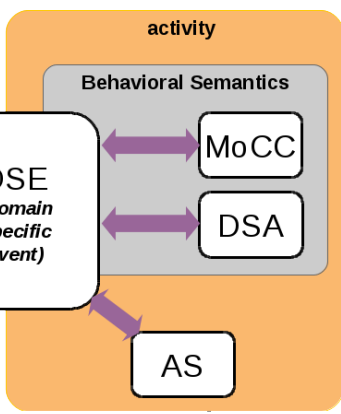
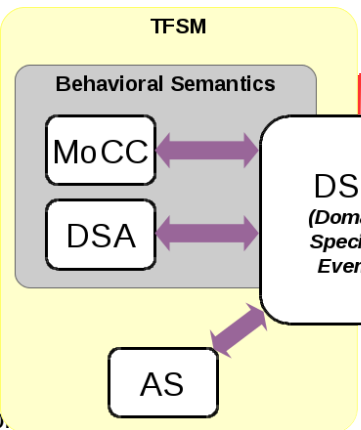
releaseCoffee:occurs

BCoOL

<<Conforms To>>

FSMEvent::occurs

Action::startAction



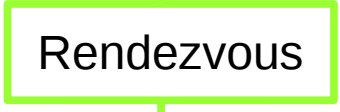
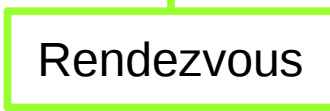
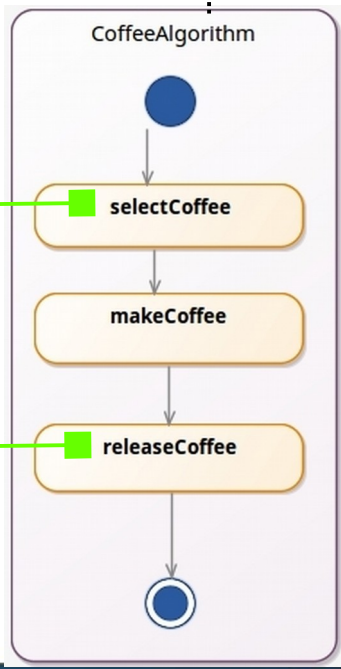
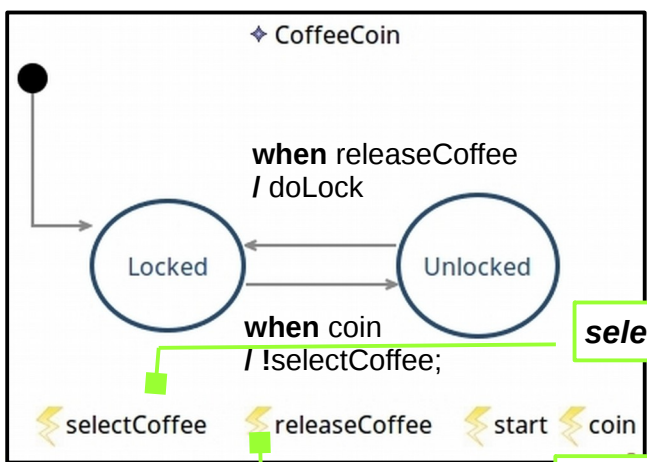
```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

Operator FSMEventsandActions
(FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```

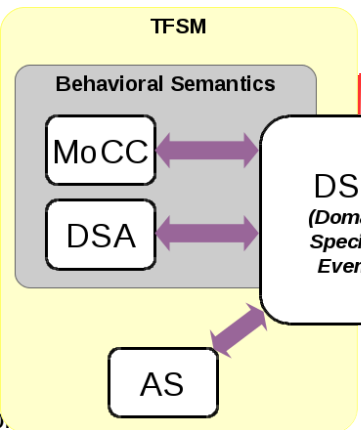
Confo.

Conforms To



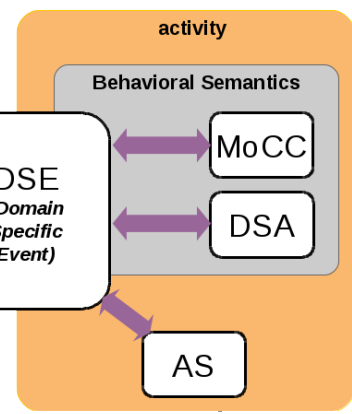
BCoOL

<<Conforms To>>



FSMEvent::occurs

Action::startAction



```

SyncFSMEventsAndActions.bcool
ImportInterface tfsm;
ImportInterface Activity;

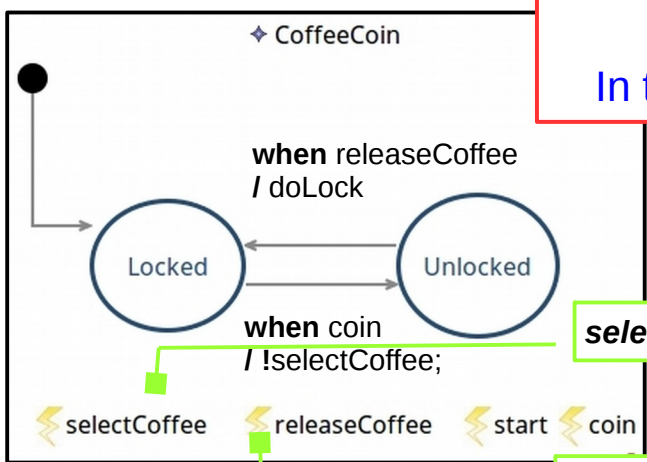
Operator FSMEventsandActions
(FSMEvent::occurs, Action::startAction)
  When(occurs.name = startAction.name);
CoordinationRule:
  Rendezvous (occurs, startAction)
End Operator;
    
```



Heterogeneous Execution In the GEMOC Studio

Rendezvous

Rendezvous

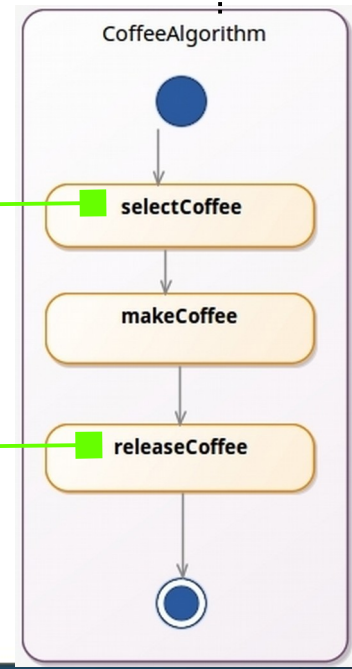


selectCoffee:occurs

releaseCoffee:occurs

selectCoffee:startAction

releaseCoffee:startAction



Implemented into the GEMOC studio

```

syncproducttfsminwithfuml.bcool
SyncProductTfsmwithfUML

ImportLib "platform:/resource/org.gemoc.bcool.example.productfumlantfsm/operator/facilities.bcoollib"
ImportLib "platform:/resource/org.gemoc.bcool.example.productfumlantfsm/operator/bcoollib.ccsllib"

ImportInterface "platform:/plugin/org.modelexecution.operationalsemantics.gemoc.ecl/model/ActivitiyDiagramV2.ecl" as ad
ImportInterface "platform:/plugin/org.gemoc.sample.tfsm.eclmoc2as/ecl/TFSM.ecl" as tfsm

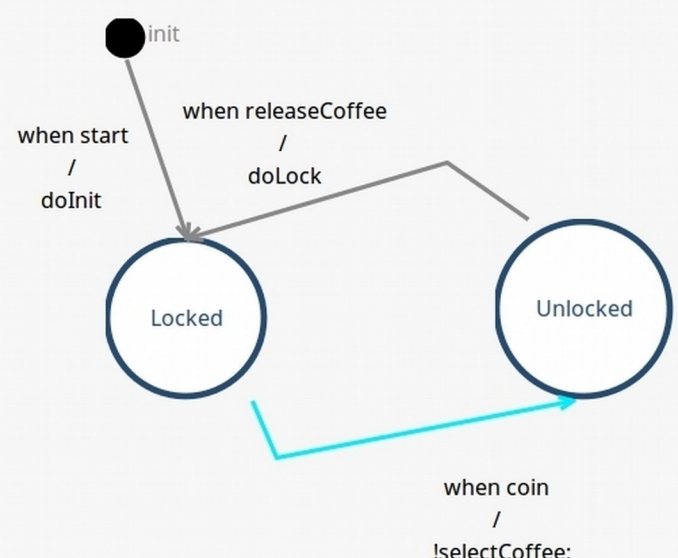
Spec test

Operator MatchingandCoordinationSharedEvents (dse1 : tfsm::occurs, dse2 : ad::executeIt)
MatchingCorrespondance: when dse1.name = dse2.name ;
CoordinationRule:
    facilities.RendezVous(dse1, dse2)
end operator;
        
```

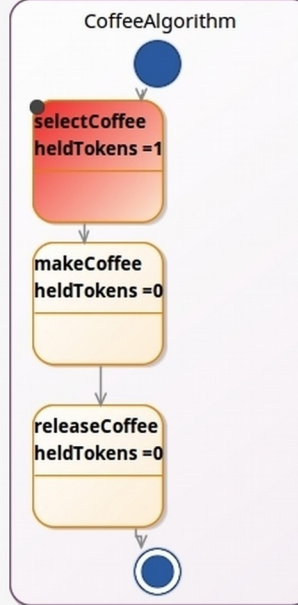
Concurrent Logical Steps Decider

- LogicalStep [903192193]
 - MSE_Unlocked_entering
 - MSE_makeCoffee_executeIt
- LogicalStep [837141265]
 - MSE_Unlocked_entering
- LogicalStep [77004704]
 - MSE_localclk_ticks
 - MSE_makeCoffee_executeIt
- LogicalStep [482013118]
 - MSE_localclk_ticks
- LogicalStep [92429118]
 - MSE_makeCoffee_executeIt
- LogicalStep [442559956]
 - MSE_Unlocked_entering
 - MSE_localclk_ticks
 - MSE_makeCoffee_executeIt
- LogicalStep [691567303]
 - MSE_Unlocked_entering
 - MSE_localclk_ticks


*CoffeeCoin



*com



Editing facilities by using Xtext

 MPM4CPS 2018 in Riga

Implemented into the GEMOC studio

```

syncproducttfsmithfuml.bcoul
SyncProductTfsmwithfUML

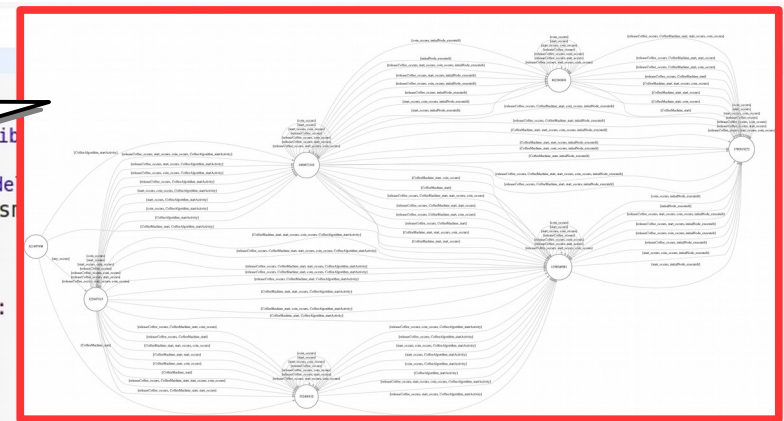
ImportLib "platform:/resource/or
ImportLib "platform:/resource/or

ImportInterface "platform:/plug
ImportInterface "platform:/plug

...
    when start
    /
    doInit
    when releaseCoffee
    /
    doLock
    when coin
    /
    !selectCoffee;

```

Schedule space exploration



Coordinated Heterogeneous Execution

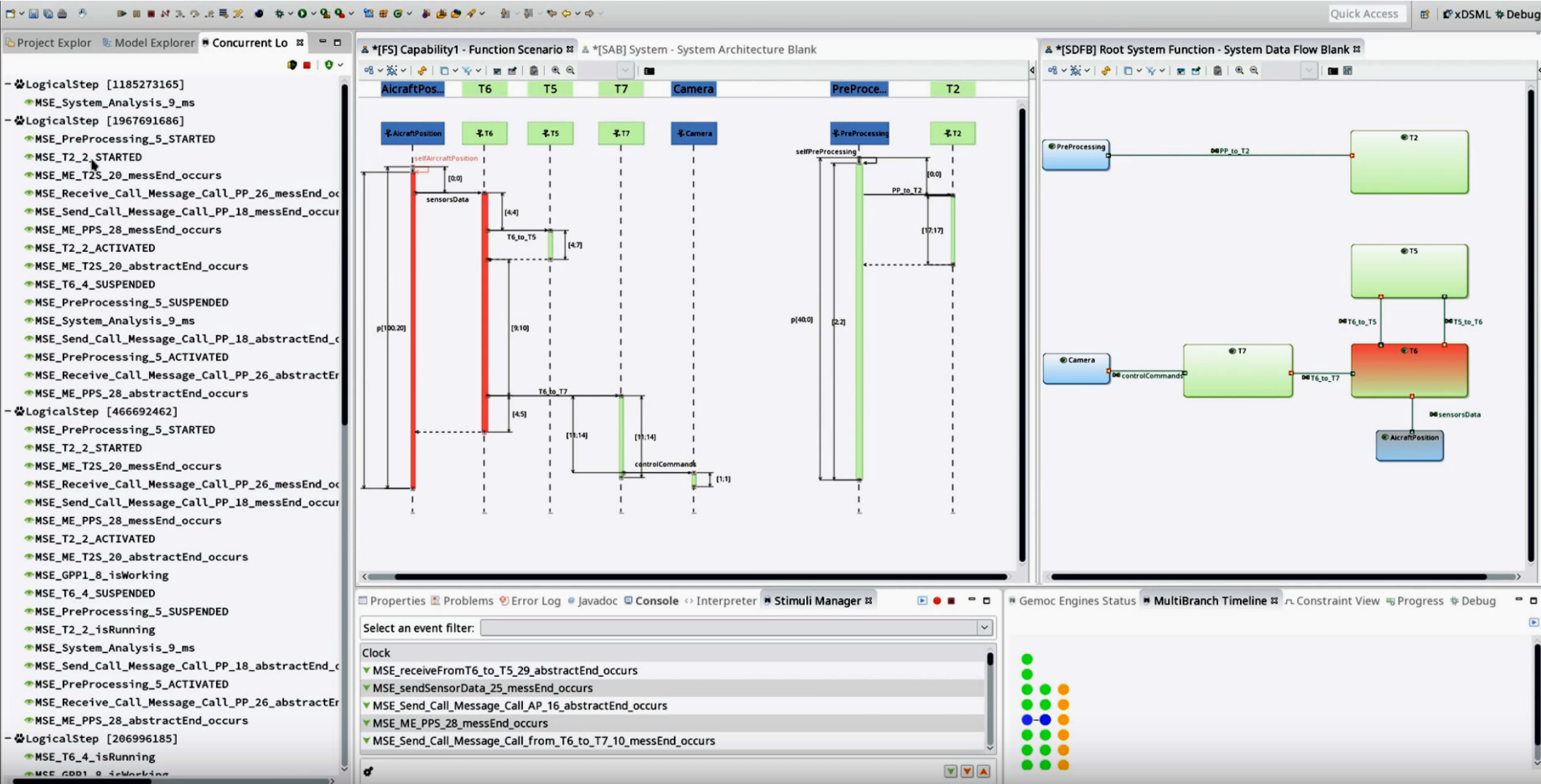
Logical Steps Decider

- LogicalStep [903192193]
 - MSE_Unlocked_entering
 - MSE_makeCoffee_executeIt
- LogicalStep [837141265]
 - MSE_Unlocked_entering
- LogicalStep [77004704]
 - MSE_localclk_ticks
 - MSE_makeCoffee_executeIt
- LogicalStep [482013118]
 - MSE_localclk_ticks
- LogicalStep [92429118]
 - MSE_makeCoffee_executeIt
- LogicalStep [442559956]
 - MSE_Unlocked_entering
 - MSE_localclk_ticks
 - MSE_makeCoffee_executeIt
- LogicalStep [691567303]
 - MSE_Unlocked_entering
 - MSE_localclk_ticks

*CoffeeCoin

*computeValueBis Activity Diagram

Modeling workbench: <http://gemoc.org/studio/>



The screenshot displays the Gemoc Studio interface with three main panels:

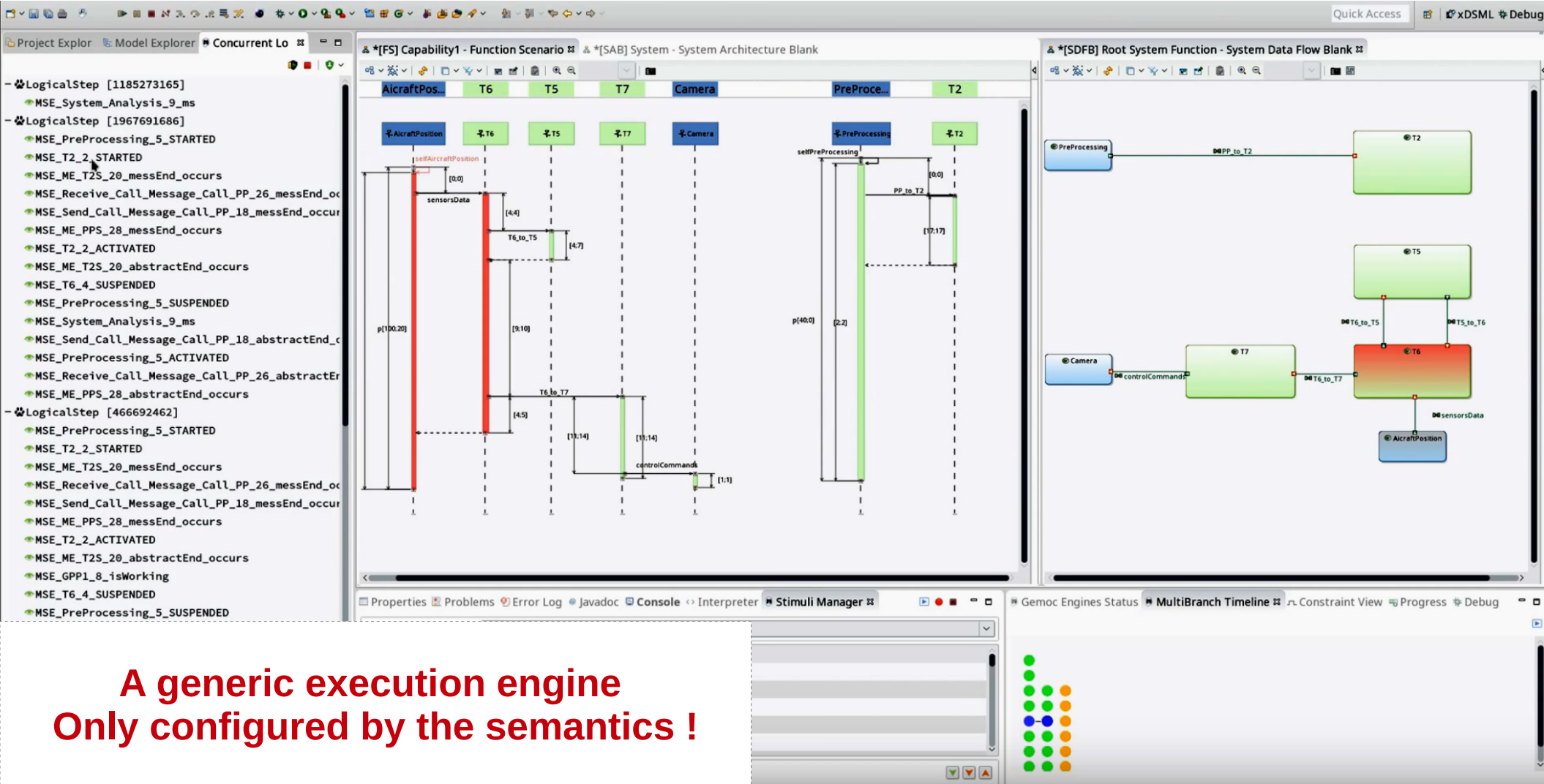
- Left Panel (Event List):** A list of events for a function scenario, including logical steps and message end occurrences (e.g., `MSE_System_Analysis_9_ms`, `MSE_PreProcessing_5_STARTED`, `MSE_T2_2_STARTED`, etc.).
- Center Panel (Sequence Diagram):** A UML sequence diagram for the function scenario. Lifelines include `AircraftPosition`, `T6`, `T5`, `T7`, `Camera`, `PreProcessing`, and `T2`. Messages shown include `selfAircraftPosition`, `sensorsData`, `T6_to_T5`, `PP_to_T2`, `T6_to_T7`, and `controlCommands`. Time intervals are marked with values like `[0:0]`, `[4:4]`, `[9:10]`, etc.
- Right Panel (Data Flow Diagram):** A system data flow diagram showing components `PreProcessing`, `T2`, `T5`, `Camera`, `T7`, `T6`, and `AircraftPosition`. Data flows include `PP to T2`, `controlCommand`, `T6_to_T7`, `T6_to_T5`, `T5_to_T6`, and `sensorsData`.

At the bottom, there are panels for **Properties**, **Stimuli Manager**, **Gemoc Engines Status**, **MultiBranch Timeline**, **Constraint View**, **Progress**, and **Debug**. The **Stimuli Manager** shows a list of events filtered by clock, such as `MSE_receiveFromT6_to_T5_29_abstractEnd_occurs`.



Making the Capella language executable....

Modeling workbench: <http://gemoc.org/studio/>



The screenshot displays the Gemoc Studio interface with three main panels:

- Left Panel (Event Log):** Lists execution events such as `MSE_System_Analysis_9_ms`, `MSE_PreProcessing_5_STARTED`, `MSE_T2_2_STARTED`, and `MSE_PreProcessing_5_SUSPENDED`.
- Middle Panel (Sequence Diagram):** Shows a sequence diagram with participants `AircraftPosition`, `T6`, `T5`, `T7`, `Camera`, `PreProcessing`, and `T2`. Messages include `selfAircraftPosition`, `sensorsData`, `controlCommands`, and `PP to T2`. Time intervals are marked with values like `[0;0]`, `[4;4]`, and `[17;17]`.
- Right Panel (Data Flow Diagram):** Shows a data flow diagram with components `PreProcessing`, `T2`, `T5`, `Camera`, `T7`, `T6`, and `AircraftPosition`. Data flows are labeled with names like `PP to T2`, `controlCommand`, `T6 to T7`, and `sensorsData`.

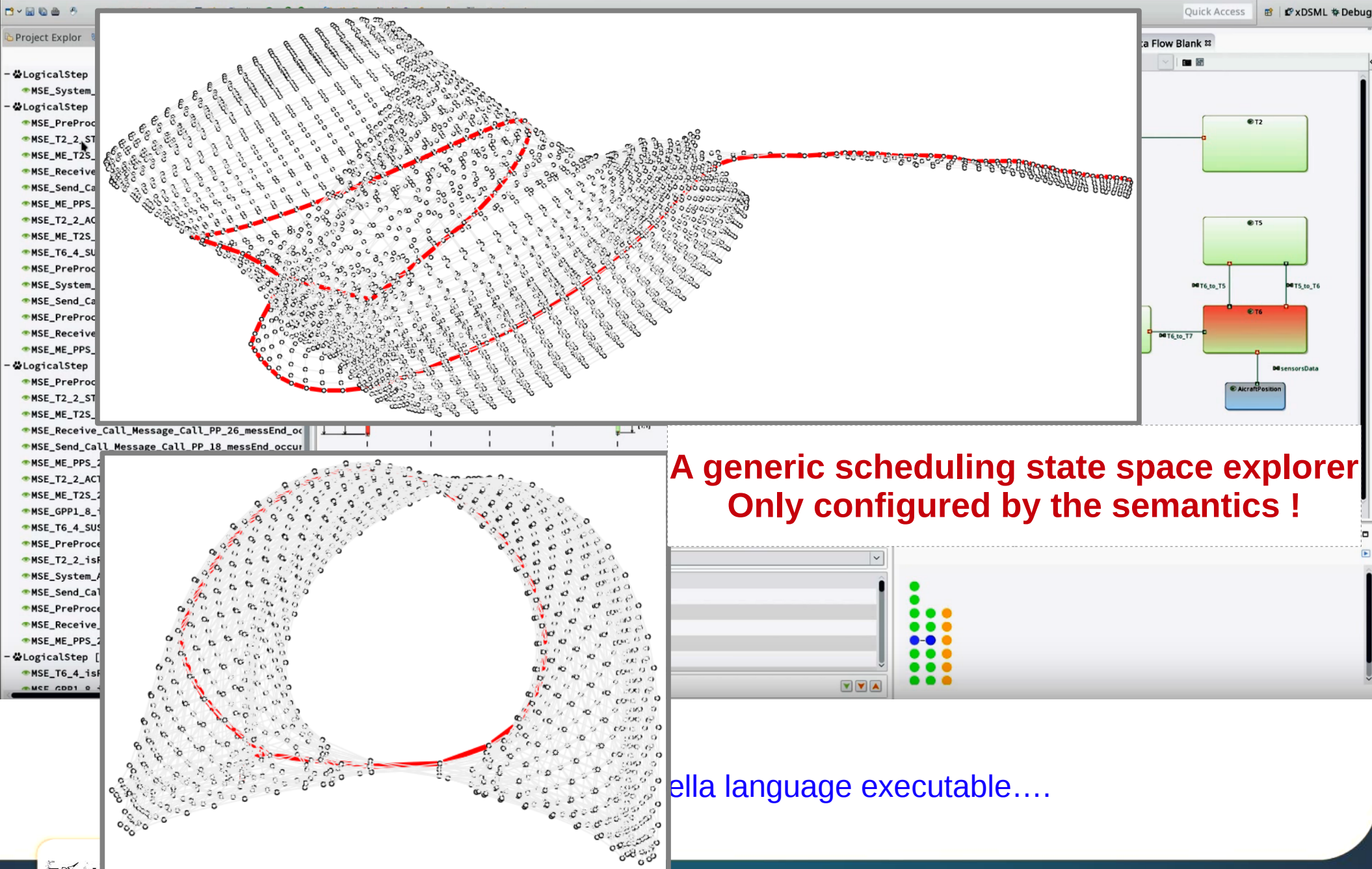
At the bottom, there are panels for `Properties`, `Problems`, `Error Log`, `Javadoc`, `Console`, `Interpreter`, `Stimuli Manager`, `Gemoc Engines Status`, `MultiBranch Timeline`, `Constraint View`, `Progress`, and `Debug`.

**A generic execution engine
Only configured by the semantics !**



Making the Capella language executable....

Modeling workbench: <http://gemoc.org/studio/>



The screenshot displays the Gemoc Studio interface. On the left, a 'Project Explorer' lists various logical steps and messages. The main area shows a state space explorer with a large, complex graph of states and transitions, highlighted with red dashed lines. On the right, a Petri net diagram shows three transitions (T2, T5, T6) and a place (AircraftPosition). The bottom right corner features a control panel with a grid of colored dots (green, blue, orange) and a vertical slider.

**A generic scheduling state space explorer
Only configured by the semantics !**

ella language executable....

Tested on several languages

- ArduinoML
- UML
 - Sequence Diagram
 - State Charts
 - Activity diagram
(challenge winner)
- Capella
 - Temporal scenario, functional architecture and deployment
 - Mode automata and functional architecture
- Deployed temporal and modal MSC
- MuVArch
- SMcube
- Communicating FSM
- TimedFSM
- Inra DSL for cow management
- ComponentModel and TFSM
- HW language for bus analysis
- MarkedGraph
- SDF (+deployment)
- AADL
- Communicating real time tasks
- ArduinoML and Sequence Diagram
- EAST-ADL

Video:

- Simulation and trace checking on the Capella system engineering language: <https://www.youtube.com/watch?v=ESIX2PFGiDU>
- Simulation of Multi communication Arduino boards: <https://youtu.be/dtJZyK1RM2A>
- BCOoL: <https://youtu.be/-o6DAzlgIMw>

Conclusion

<http://gemoc.org>

- We provided meta languages for:
 - The modeling of executable languages
 - The modeling of coordination patterns
- We tooled the meta languages so that;
 - A modeling workbench with debugging facilities is generated from the language specification
 - An heterogeneous execution engine is generated from a coordination pattern and specific models
 - Generic analysis is possible
- We are currently:
 - Applying such techniques to study interaction in model-based system engineering solutions (with Thales and Safran)
 - Extending the approach to support continuous time models
 - Studying how this co modeling can be used to generate co simulation masters (based on FMI standard)
 - Transferring the technology in a company that uses Connected Objects and IoT standard to monitor senior daily live activities environment in a non intrusive way.

Conclusion

<http://gemoc.org>

- We provided meta languages for:
 - The modeling of executable languages
 - The modeling of coordination patterns
- We tooled the meta languages so that;
 - A modeling workbench with debugging facilities is generated from the language specification
 - An heterogeneous execution engine is generated from a coordination pattern and specific models
 - Generic analysis is possible
- We are currently:
 - Applying such techniques to study interaction in model-based system engineering solutions (with Thales and Safran)
 - Extending the approach to support continuous time models
 - Studying how this co modeling can be used to generate co simulation masters (based on FMI standard)
 - Transferring the technology in a company that uses Connected Objects and IoT standard to monitor senior daily live activities environment in a non intrusive way.

Conclusion

<http://gemoc.org>

- We provided meta languages for :
 - The modeling of executable languages
 - The modeling of coordination patterns
- We tooled the meta languages so that :
 - A modeling workbench with debugging facilities is generated from the language specification
 - An heterogeneous execution engine is generated from a coordination pattern and specific models
 - Generic analysis is possible
- We are currently:
 - Applying such techniques to study interaction in model-based system engineering solutions (with Thales and Safran)
 - Extending the approach to support continuous time models
 - Studying how this co modeling can be used to generate co simulation masters (based on FMI standard)
 - Transferring the technology in a company that uses Connected Objects and IoT standard to monitor senior daily live activities in a non intrusive way.