

MODEL EXECUTION

BUILD YOUR OWN COMPILER AND VIRTUAL MACHINE

ESIR3 ASE, 2022-2023

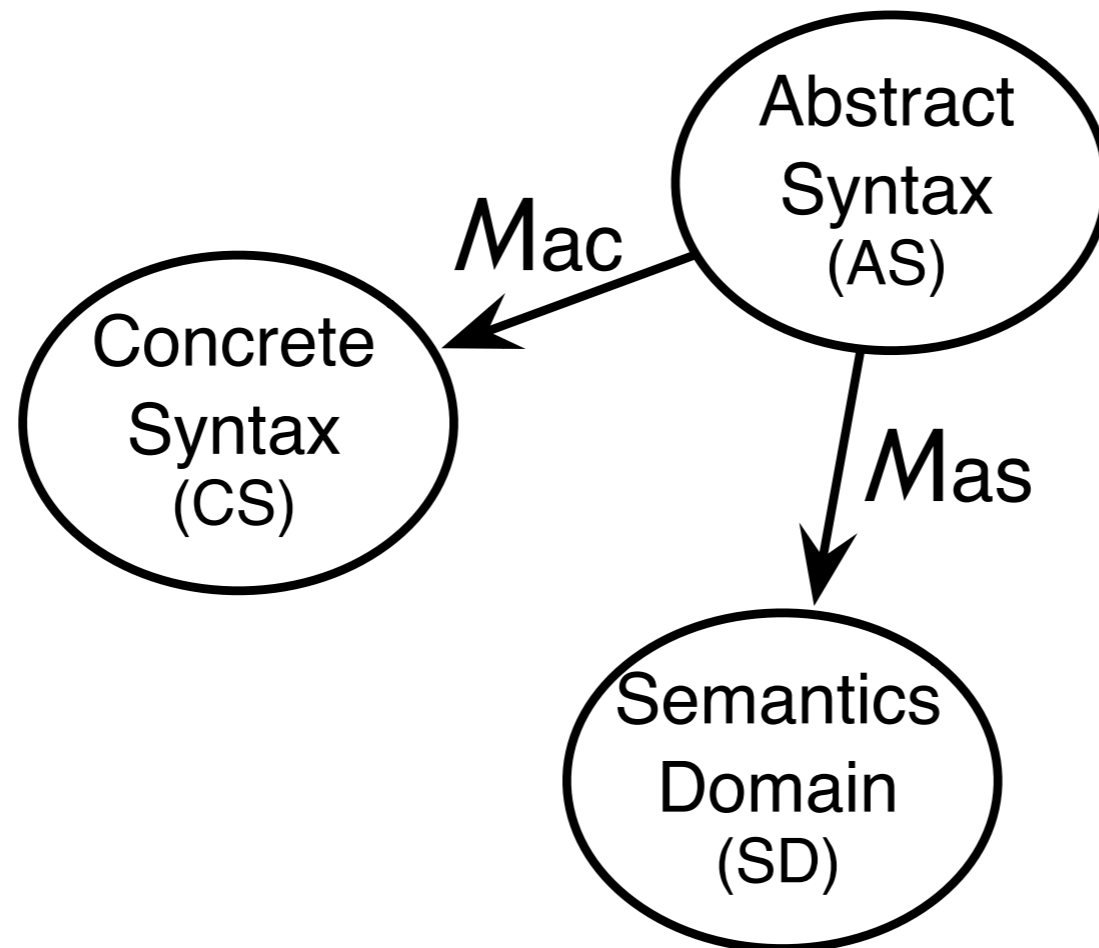
BENOIT COMBEMALE

PROFESSOR, UNIV. RENNES 1 & INRIA, FRANCE

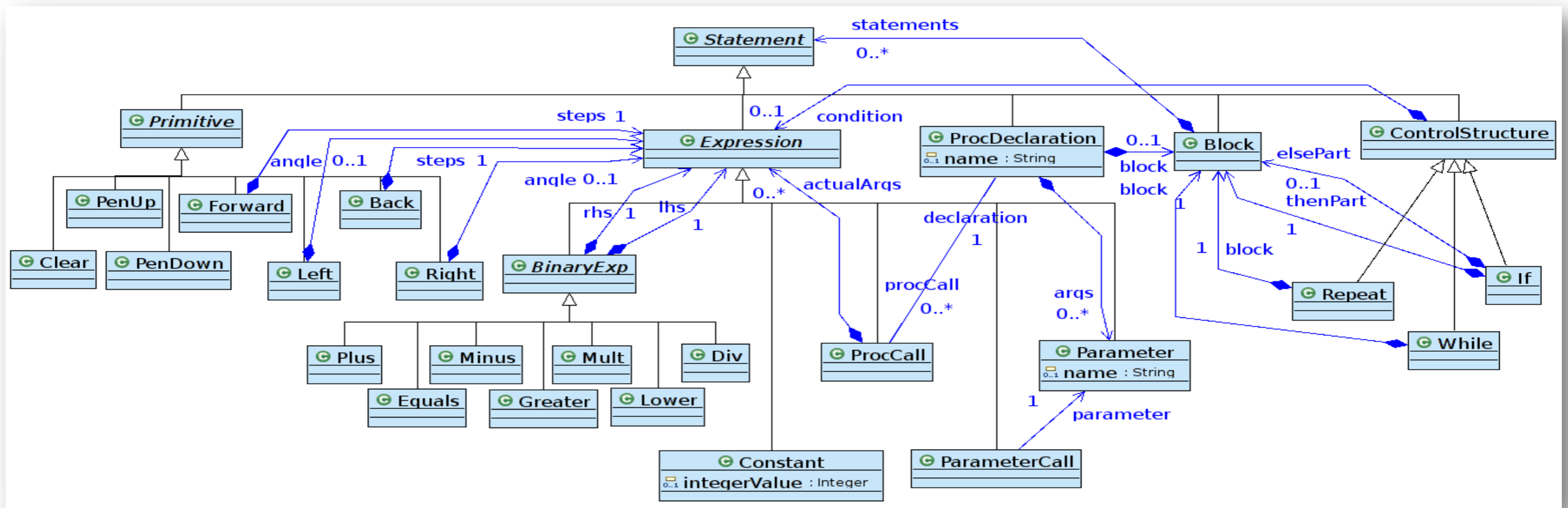
[HTTP://COMBEMALE.FR](http://combemale.fr)
[BENOIT.COMBEMALE@IRISA.FR](mailto:benoit.combemale@irisa.fr)
[@BCOMBEMALE](https://twitter.com/bcombemale)



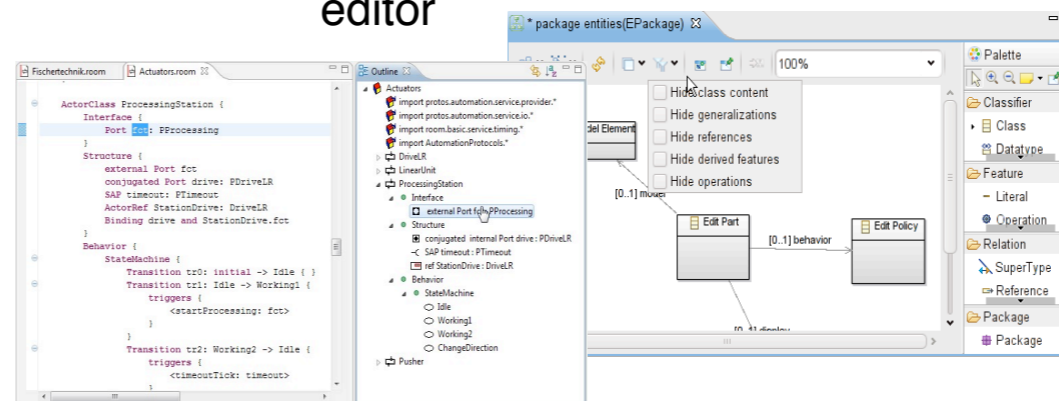
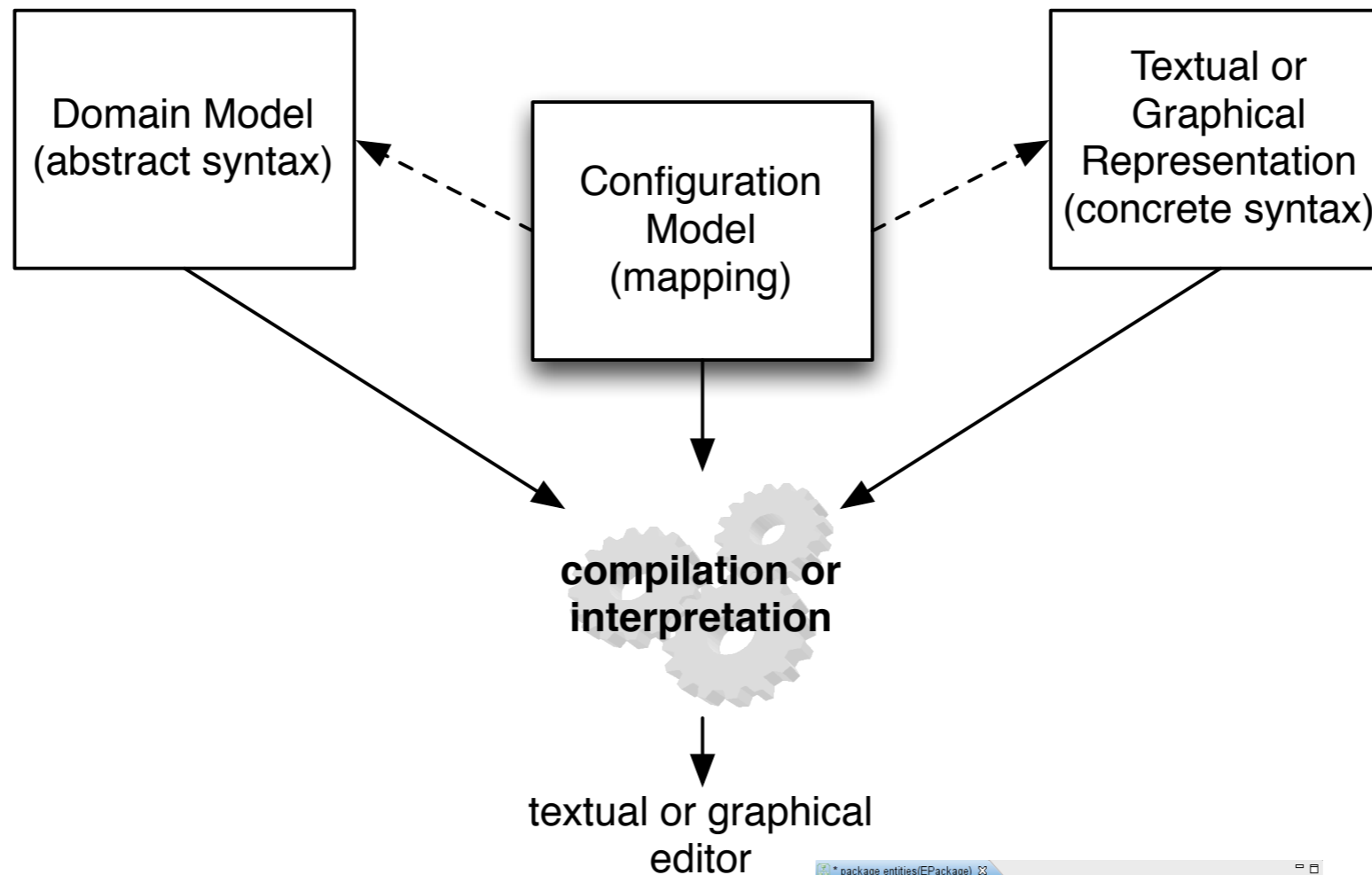
Reminder about what is a language



Reminder about what is an abstract syntax



Reminder about what is a concrete syntax



Reminder about what is a semantics

- ▶ Any “meaning” given to the domain model
 - compiler, interpreter, analysis tool, refactoring tool, etc.
- ▶ Thanks to model transformations
 - program = data + algorithms 😊
- ▶ In practices?
 - It requires to “traverse” the domain model, and... do something!
 - Various languages, and underlying paradigms:
 - *Declarative* (rule-based): mostly for pattern matching (e.g., analysis, refactoring)
 - *Imperative* (visitor-based):
 - *interpreter pattern*: mostly for model interpretation (e.g., execution, simulation)
 - *template*: mostly for text generation (e.g., code/test/doc generators)

Reminder of the previous lectures / labs

Build your own (Domain-Specific) Language

1. Build your abstract syntax as a domain model with Ecore (possibly additional constraints with OCL, aka. context conditions)
2. Build your concrete syntax (textual with Xtext, *graphical with Sirius*)
3. Build your generators
 - ▶ Documentation generator
 - ▶ Code generator (/compiler)

Objectives of the coming lecture/labs

4. Build your interpreter (/ VM)
5. Build your animator

Get your own modeling workbench with
model edition, compilation, execution,
simulation, (graphical) animation and debugging

Definition of the Behavioral Semantics of DSL

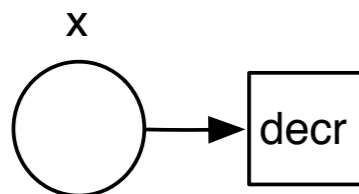
```
int x;  
void decr () {  
    if ( x>0 )  
        x = x-1;  
}
```

System
x : Int
decr()

▶ Axiomatic

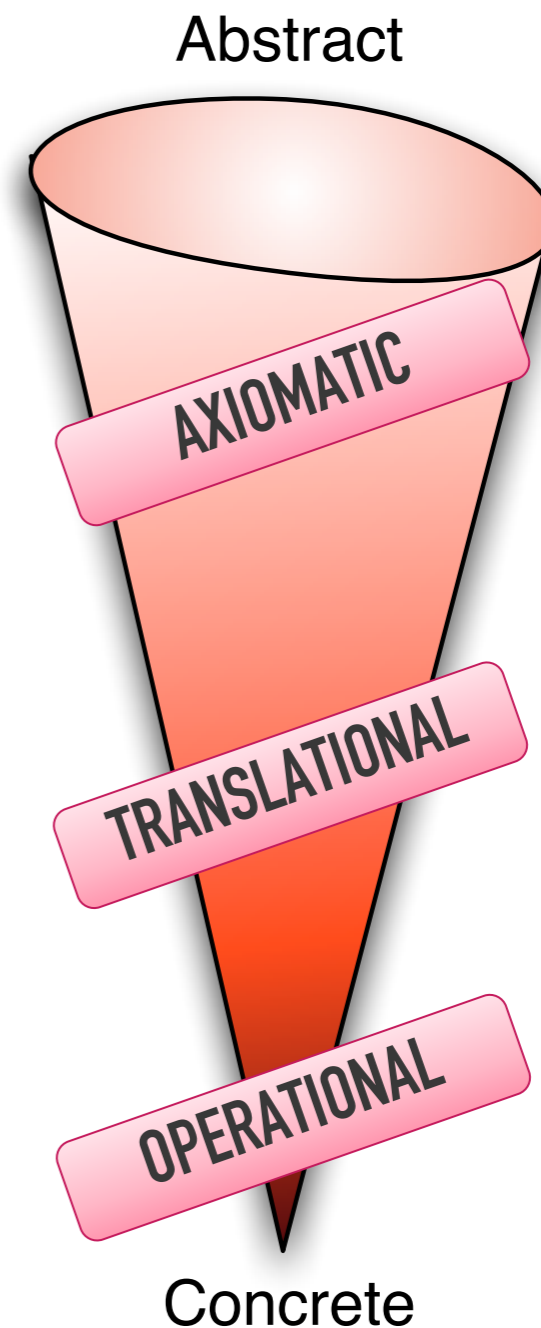
```
context System::decr() post :  
    self .x = if ( self .x@pre>0 )  
                then self.x@pre - 1  
                else self.x@pre  
            endif
```

▶ Denotational/translational

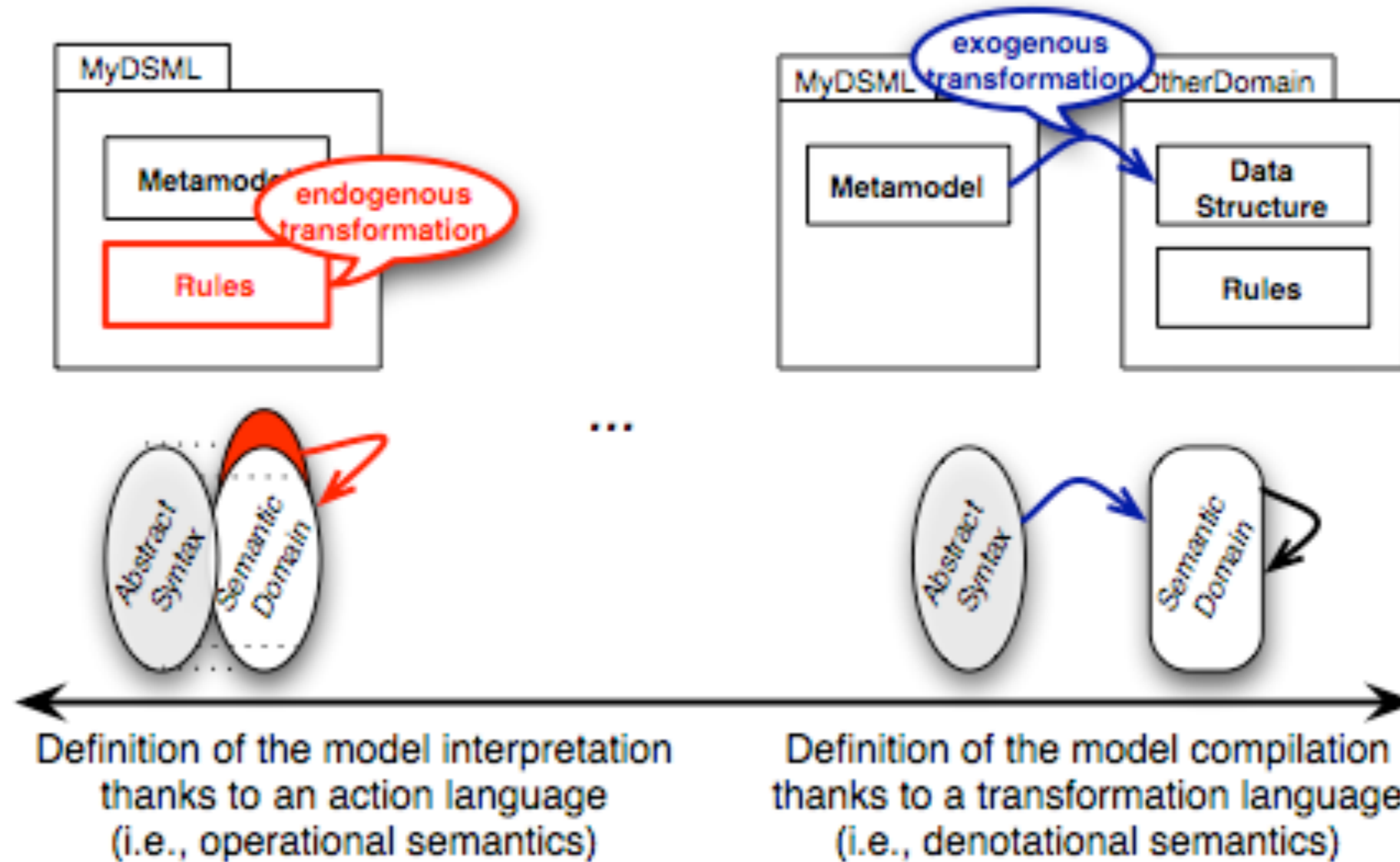


▶ Operational

```
operation decr () is do  
    if x>0 then x = x - 1 end
```



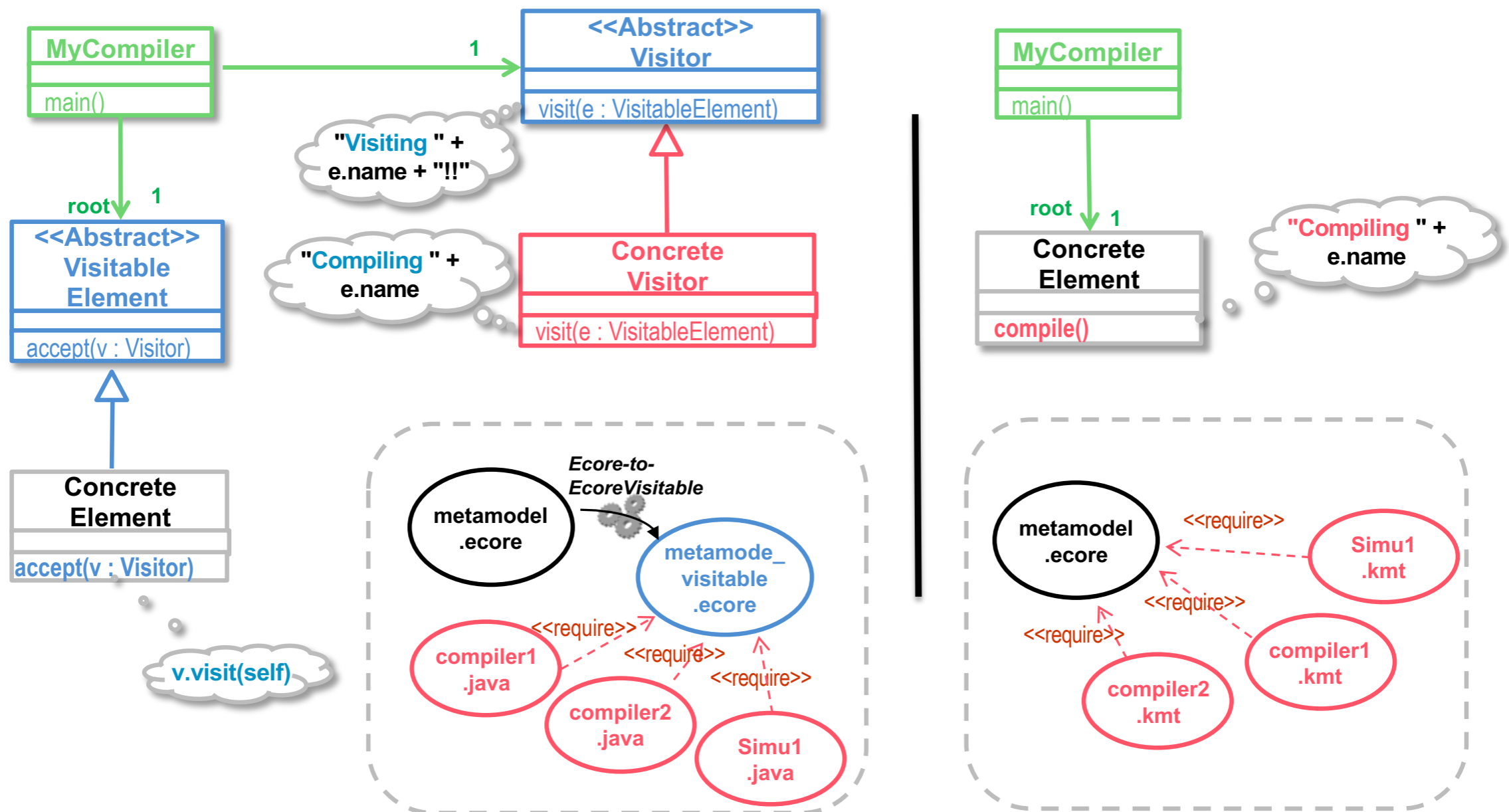
Definition of the Behavioral Semantics of DSL



Implement your own compiler / interpreter

▶ Visitor-based?

- ▶ Interpreter/visitor patterns, static introduction (aka. open class)



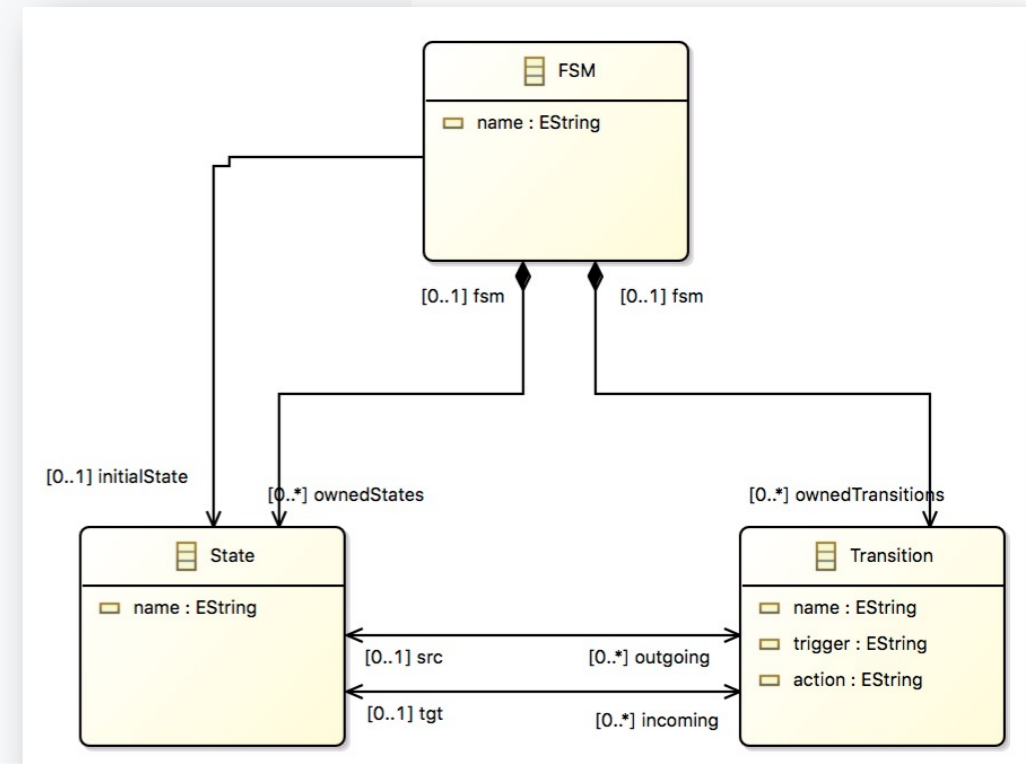
Implement your own interpreter with Xtend/K3

```
@Aspect(className=State)
class StateAspect {
    @Step
    def public void step(String inputString) {
        // Get the valid transitions
        val validTransitions = _self.outgoing.filter[t | inputString.compareTo(t.trigger) == 0]

        if(validTransitions.empty) {
            //just copy the token to the output buffer
            _self.fsm.outputBuffer.enqueue(inputString)
        }

        if(validTransitions.size > 1) {
            throw new Exception("Non Determinism")
        }

        // Fire transition first transition (could be random%VT.size)
        if(validTransitions.size > 0){
            validTransitions.get(0).fire
            return
        }
        return
    }
}
```

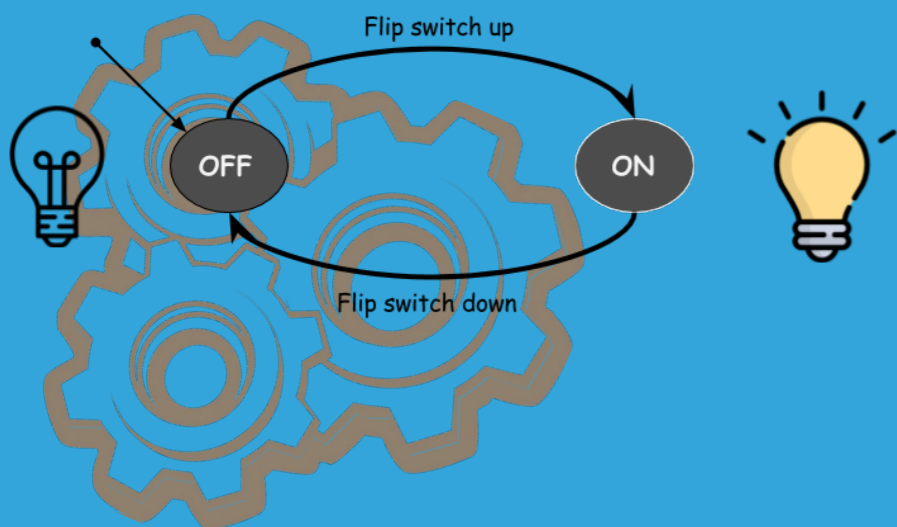


```
@Aspect(className=Transition)
class TransitionAspect {
    @Step
    def public void fire() {
        println("Firing " + _self.name + " and entering " + _self.tgt.name)
        val fsm = _self.src.fsm
        fsm.currentState = _self.tgt
        fsm.outputBuffer.enqueue(_self.action)
        fsm.consummedString = fsm.consummedString + fsm.underProcessTrigger
    }
}
```

```
@Aspect(className=FSM)
class FSMAspect {

    public State currentState
}
```

Part 4: define an interpreter for your language



KEEP

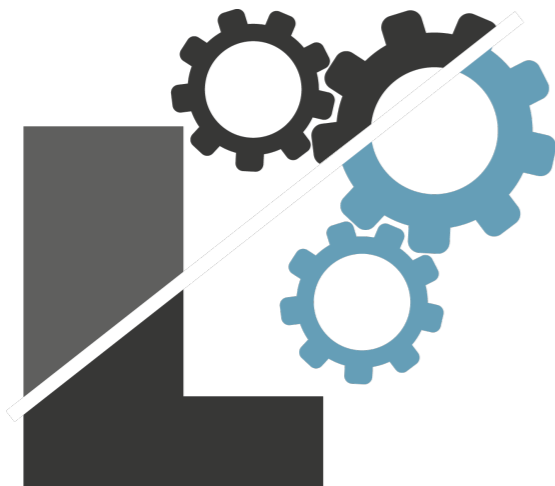
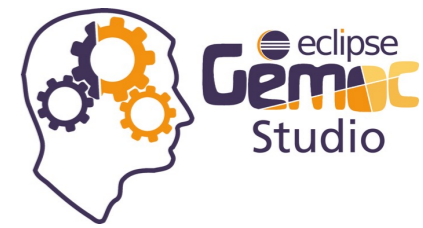
CALM

AND

DO IT

YOURSELF

The GEMOC Studio



***Language
Workbench***

*Design and integrate your
executable DSMLs*

<http://gemoc.org/studio>

and

<http://eclipse.org/gemoc>



***Modeling
Workbench***

*Edit, simulate and animate your
heterogeneous models*

Arduino Designer



The screenshot displays the Eclipse Gemoc Studio interface for Arduino Designer. The top toolbar includes navigation and execution controls. The main workspace is divided into several panels:

- Debug:** Shows the execution context for the 'blinker' project, including the Gemoc debug target and model debugging details.
- Hardware:** Displays a 3D model of an Arduino Board with three LEDs (RED, BLUE, WHITE) connected to pins 0, 1, and 2.
- Sketch:** Shows a flowchart for a 'Repeat 5' loop. The loop starts with $i = 0$ and contains three parallel LED control blocks: 'BLUE LED : $(i\%2)$ ', 'RED LED : $((i/2)\%2)$ ', and 'WHITE LED : $((i/4)\%2)$ '. Each block is connected to an 'int 2' variable. The loop concludes with a 'Set $i = (i+1)$ ' block, which is connected to an 'int 1' variable.
- Console:** Shows the modeling workbench console output: 'About to initialize and run the GEMOC Execution Engine... Initialization done, starting engine...'
- Variables:** A table showing the current state of variables during execution.

Name	Value
i (org.gemoc.sequential.model.arduino.impl.RepeatImpl@4e523b1 (iteration: 5):Repeat)	0
level (Arduino Board.0 :DigitalPin)	0
level (Arduino Board.1 :DigitalPin)	0
level (Arduino Board.2 :DigitalPin)	0
value (newSketch.i :IntegerVariable)	0

<https://github.com/gemoc/arduinomodeling>

Model Execution (ESIR)
Benoit Combemale, Nov. 2022

UML Activity Diagram



Debug - platform:/resource/org.modelexecution.operacionalsemantics.ad.samplemodels/model/test2.aird/test2 Activity Diagram - Gemoc Studio

File Edit Diagram Navigate Search Project Run Window Help

Quick Access Debug xDSML

Debug test2.ad [Gemoc Sequential eExecutable Model]

- Gemoc debug target
- Model debugging
 - (ForkNode) test2.forkNode1 -> execute()
 - (Activity) test2 -> execute()
- Global context : Activity

(x)= Variables

Name	Value
heldTokens (ActivityFinalNode_finalNode2 :ActivityFinalNode)	[]
heldTokens (ForkNode_forkNode1 :ForkNode)	[]
heldTokens (InitialNode_initialNode2 :InitialNode)	[activitydiagram.impl.ControlTokenImpl]
heldTokens (JoinNode_joinNode1 :JoinNode)	[]

Breakpoints

- Opaque Action action2

No details to display for the current selection.

Console *test2 Activity Diagram

Properties Opaque Action action2

Property	Value
Activity	test2
Incoming	Control Flow edge4
Outgoing	Control Flow edge6
Running	true

Gemoc Engines Status

- test2.ac 8

Multidimensional Timeline

All execution states (11)

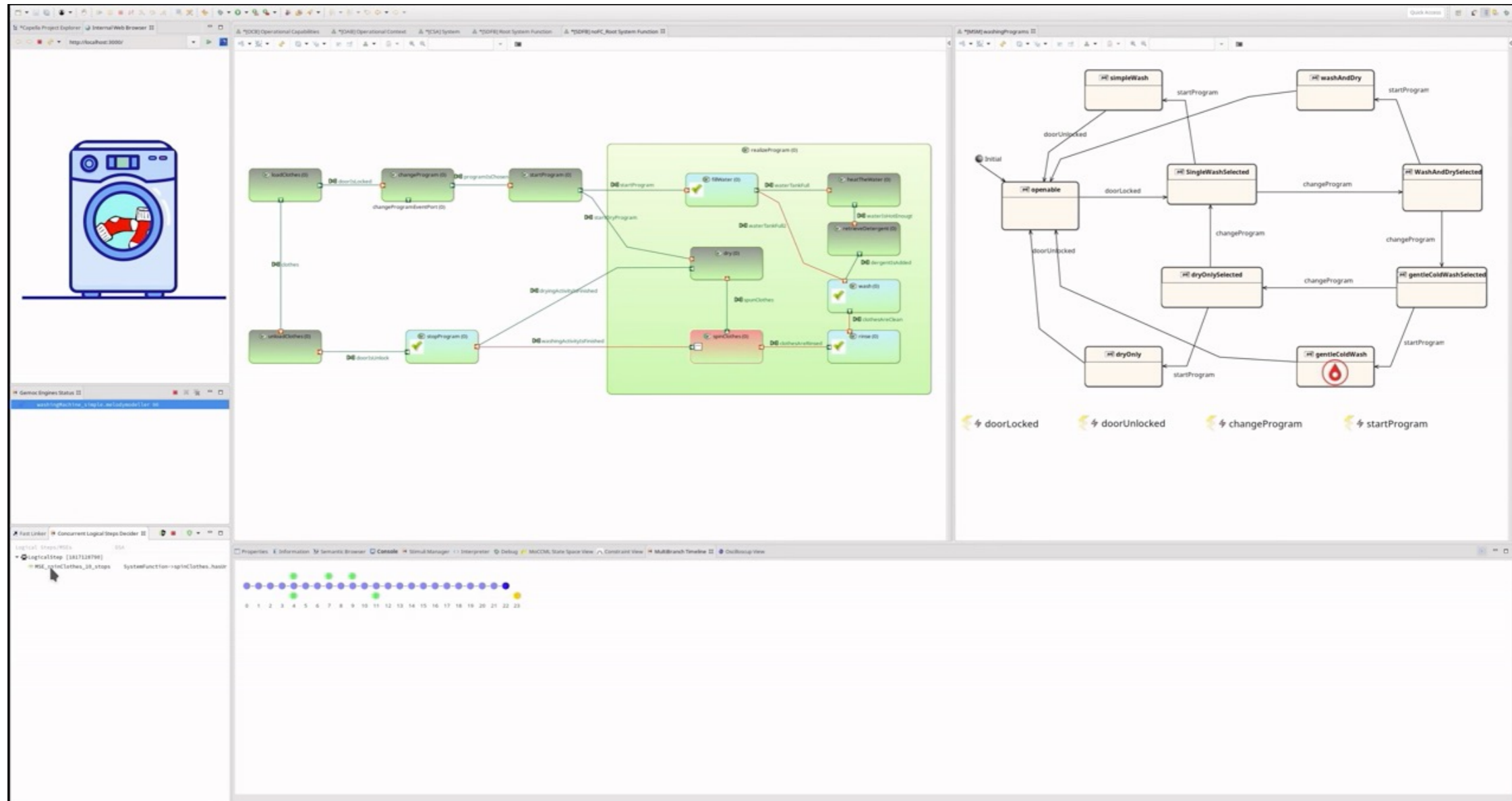
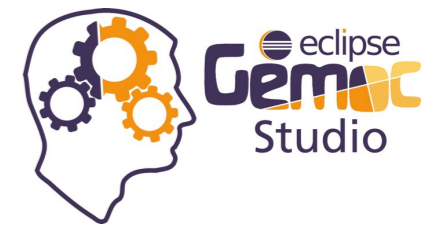
Timeline for dynamic information

trace (test2 :Activity)

- heldTokens (test2.initialNode2 :InitialNode)
- heldTokens (test2.forkNode1 :ForkNode)
- heldTokens (test2.action2 :OpaqueAction)
- heldTokens (test2.action3 :OpaqueAction)
- heldTokens (test2.joinNode1 :JoinNode)
- heldTokens (test2.finalNode2 :ActivityFinalNode)

<https://github.com/gemoc/activitydiagram>

xCapella



FCL



The screenshot displays the Eclipse Gemoc Studio interface with several windows open:

- Project Explorer:** Shows the project structure for 'MiniQuadCopter.fcl'. The 'FCL Mode Diagram' is selected.
- QuadCtrlNoFMU Function Diagram:** A complex state machine diagram with multiple states and transitions.
- MiniQuadCopter.fcl:** A code editor showing the following code:

```
1 lightControllerModel QuadCtrlNoFMU {
2   modeAutomata masterAutomata {
3     mode ManualStabilizedFlight {
4       // Stabilize mode allows you to fly your veh:
5       // http://ardupilot.org/copter/docs/stabiliz
6       // Pilot's roll and pitch input control the :
7       // Pilot will need to regularly input roll ar
8       // Pilot's yaw input controls the rate of ch
9       // Pilot's throttle input controls the avera
10      // The throttle sent to the motors is automa
11      enabledFunctions {
12        masterFunction.FlightController. // Do I
13        masterFunction.FlightController.getDesir
14        masterFunction.FlightController.getDesir
15        masterFunction.FlightController.getDesir
16        masterFunction.FlightController.getStabi
17        masterFunction.Environment.
18        masterFunction.VirtualSensors
19      }
20    }
21  }
22 }
```
- QuadCtrlNoFMU Mode Diagram:** A state machine diagram with states like 'ManualStabilizedFlight' and transitions based on events like 'RCCommandReceived'.
- Context Menu:** A menu is open over the mode diagram with options: Edit, Show/Hide, Layout, Reset Origin, Format, Show EClass information, Show References, OCL, and Open in textual editor.
- Console:** Shows a log entry: '14:48:28.584 [Worker-3: Decoration Calcula...]git.util.FS - remaining o...'