

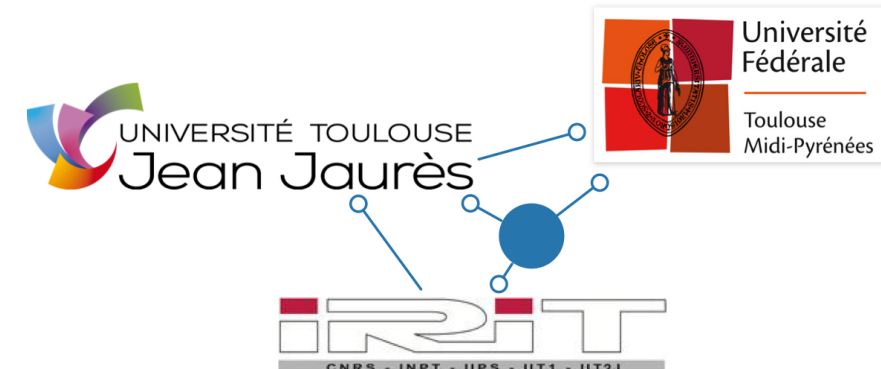
MODEL EXECUTION

BUILD YOUR OWN VM FOR YOUR LANGUAGE

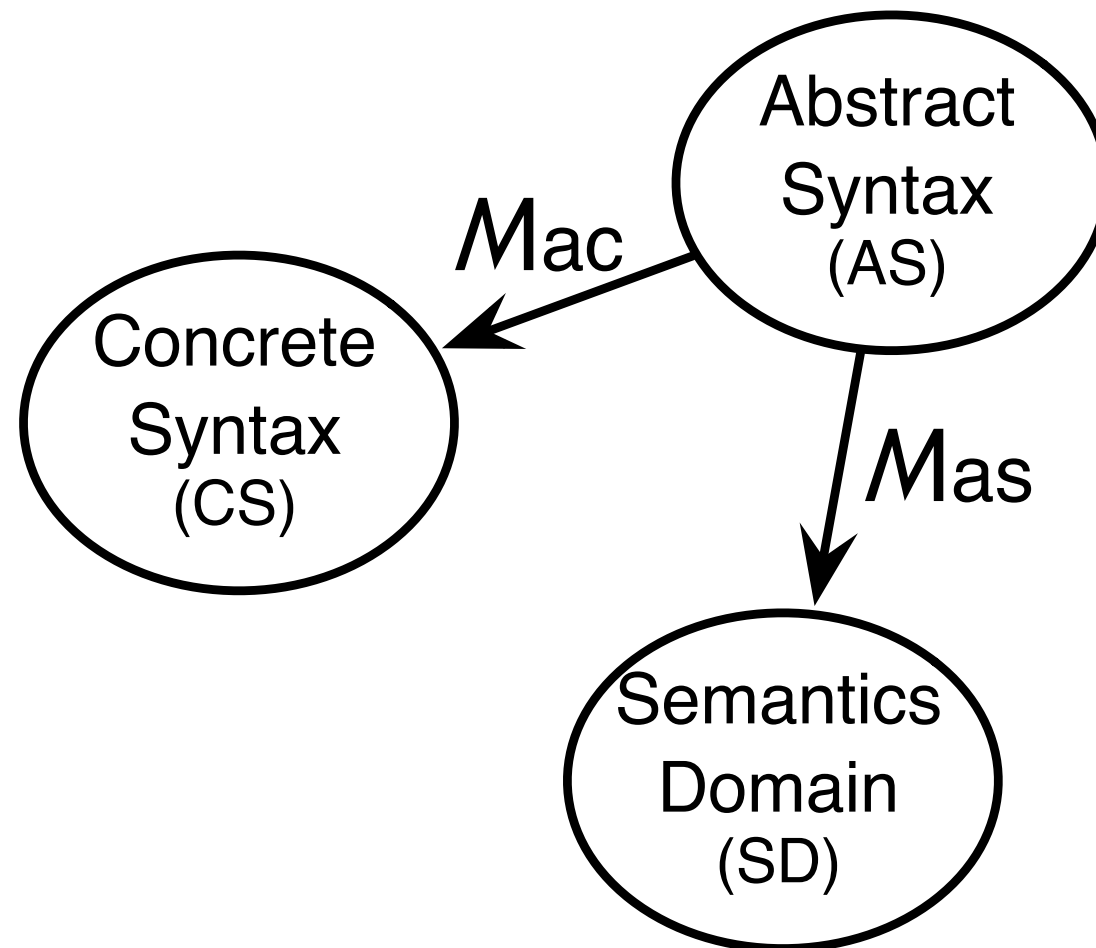
MASTER 1 ICE, 2017-2018

BENOIT COMBEMALE
PROFESSOR, UNIV. TOULOUSE, FRANCE

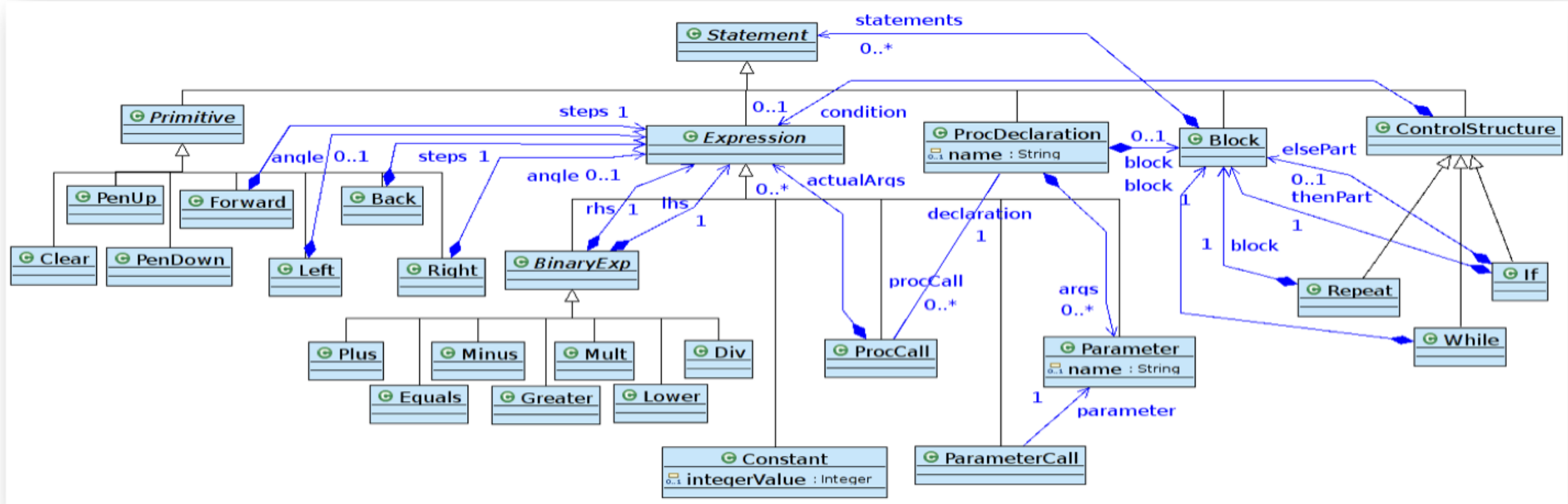
[HTTP://COMBEMALE.FR](http://combemale.fr)
[BENOIT.COMBEMALE@IRIT.FR](mailto:benoit.combemale@irit.fr)
[@BCOMBEMALE](https://twitter.com/BCOMBEMALE)



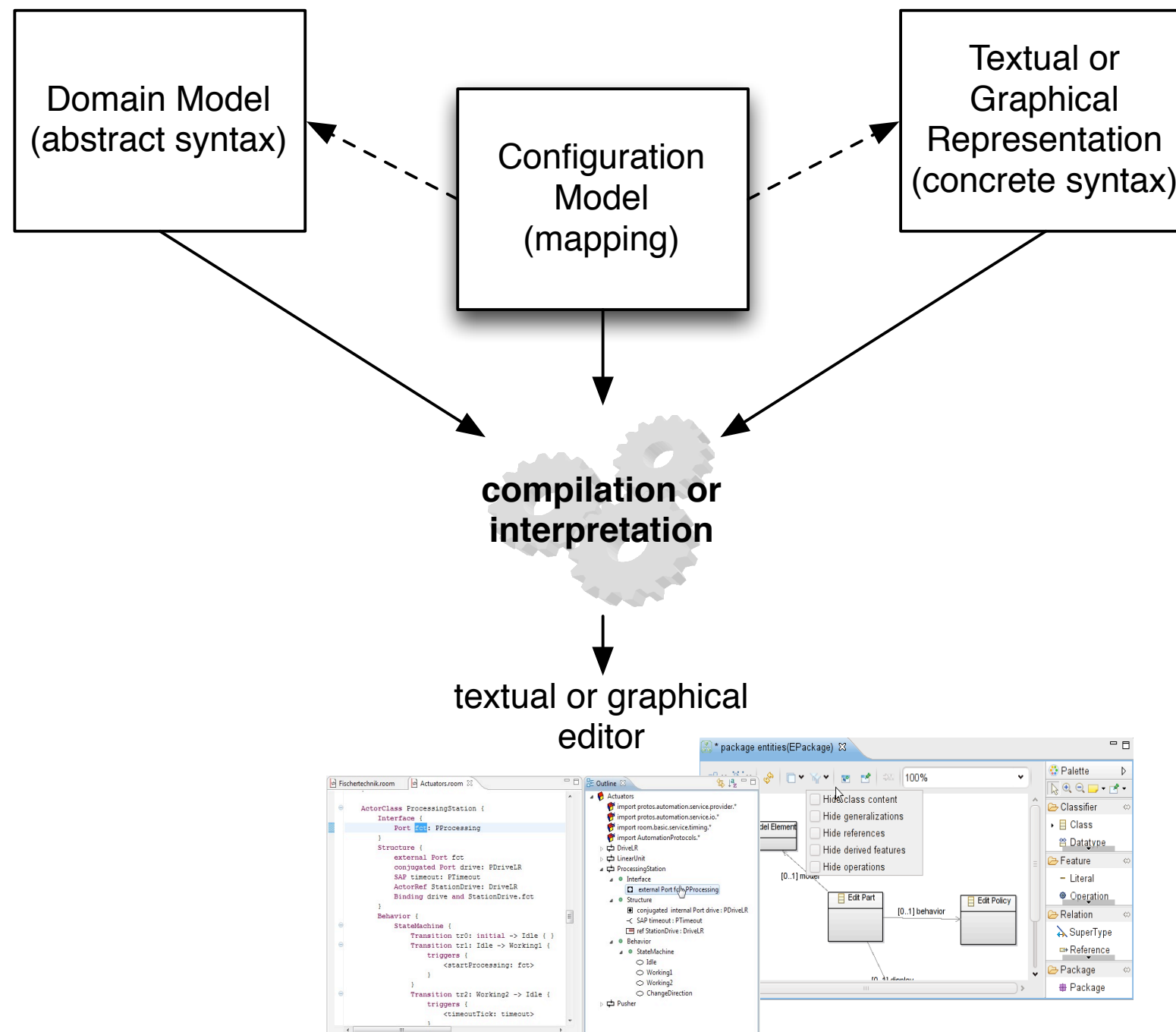
Reminder about what is a language



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

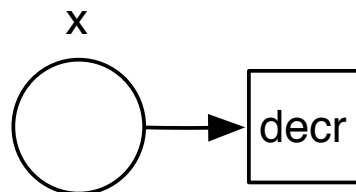
```
int x;  
void decr () {  
    if ( x>0 )  
        x = x-1;  
}
```

System
x : Int
decr()

► Axiomatic

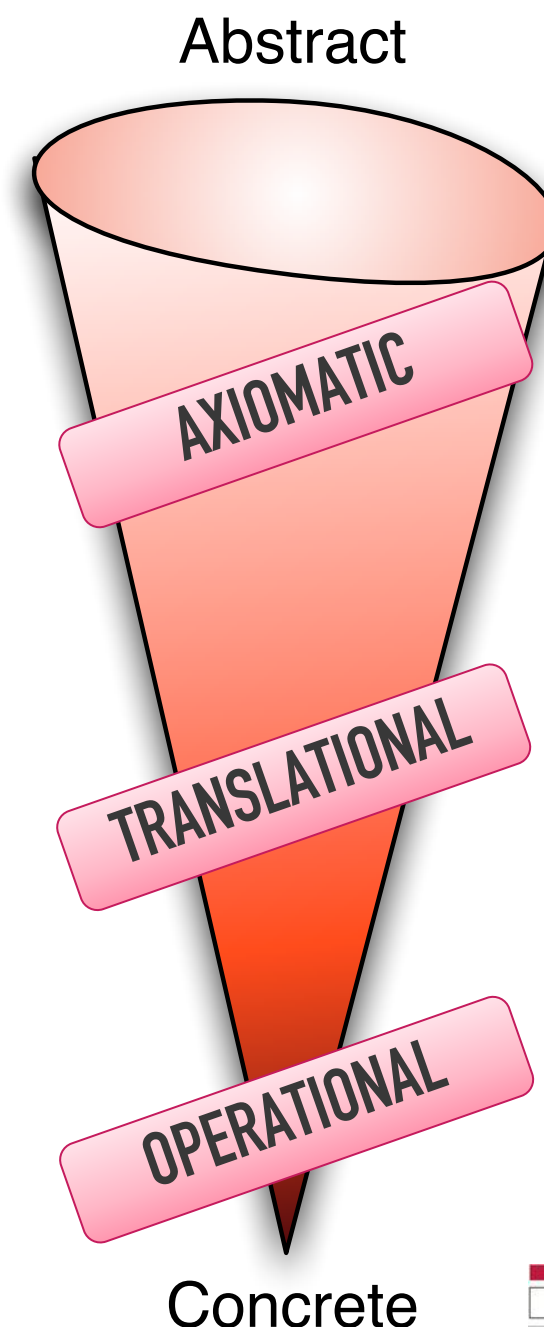
```
context System::decr() post :  
    self.x = if ( self.x@pre>0 )  
                then self.x@pre - 1  
                else self.x@pre  
            endif
```

► Denotational/translational

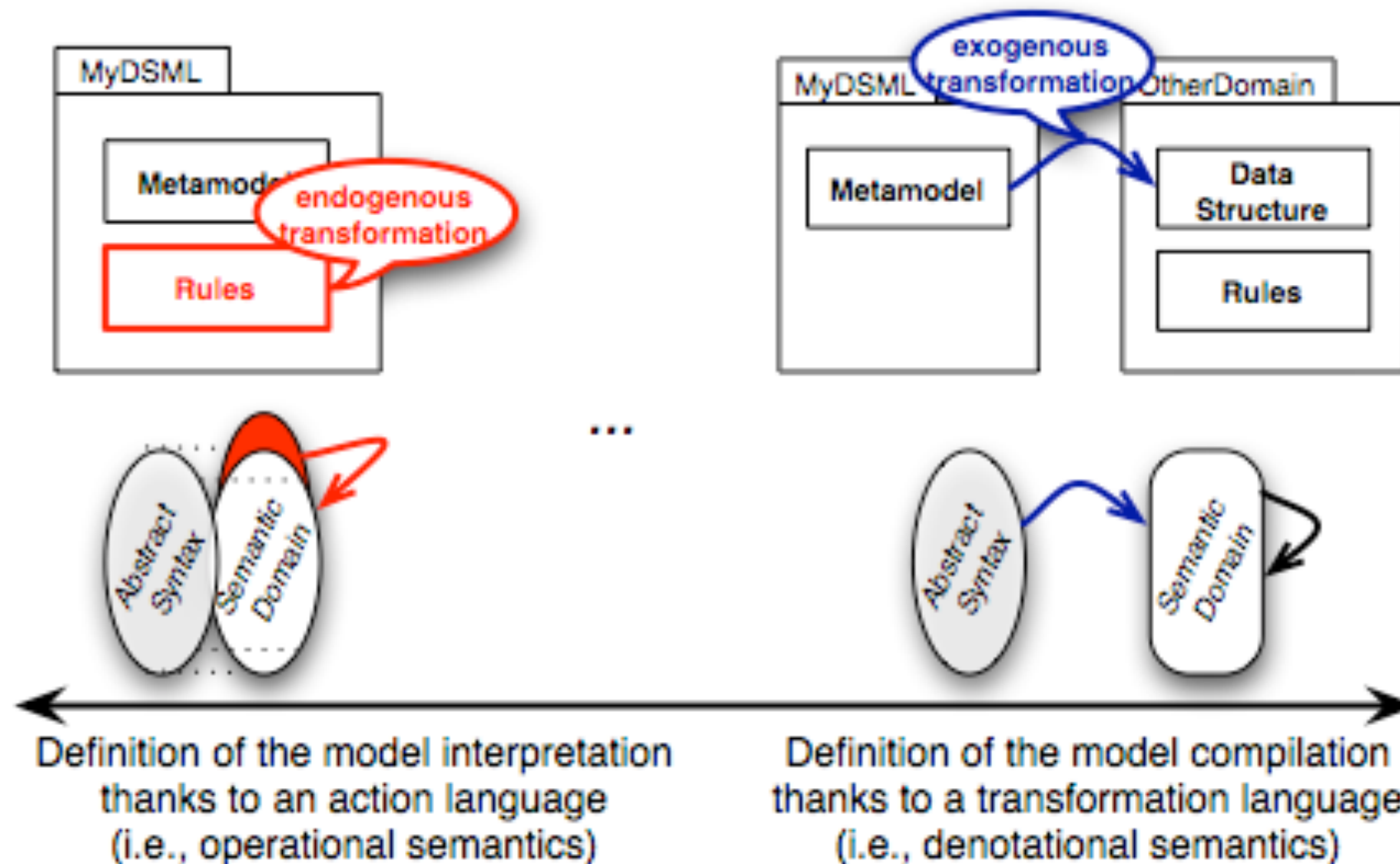


► Operational

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

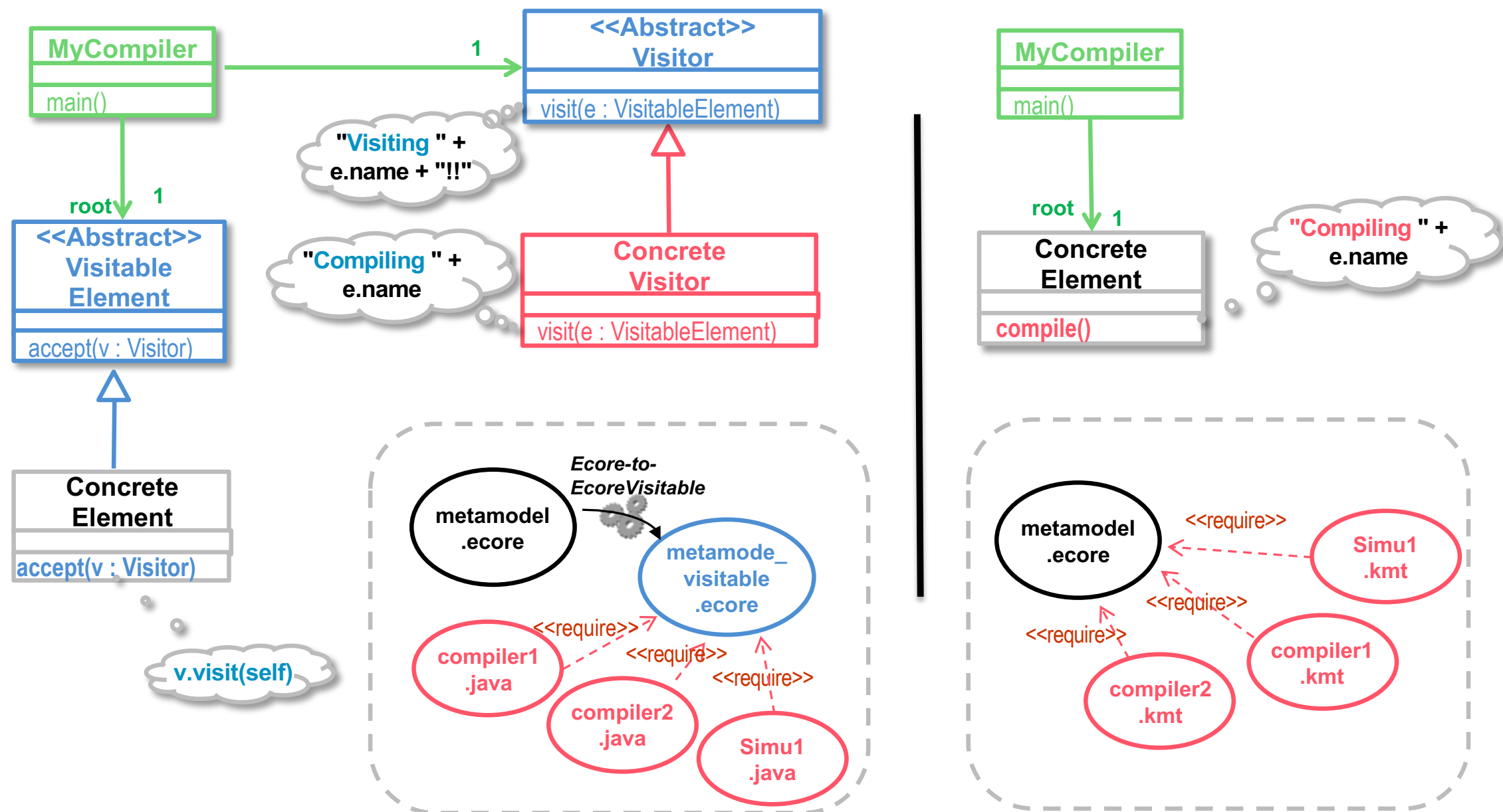


Definition of the Behavioral Semantics of DSL



Implement your own interpreter

- ▶ Visitor-based?
 - ▶ Interpreter pattern, static introduction (aka. open class)



Implement your own interpreter with Xtend/K3

```

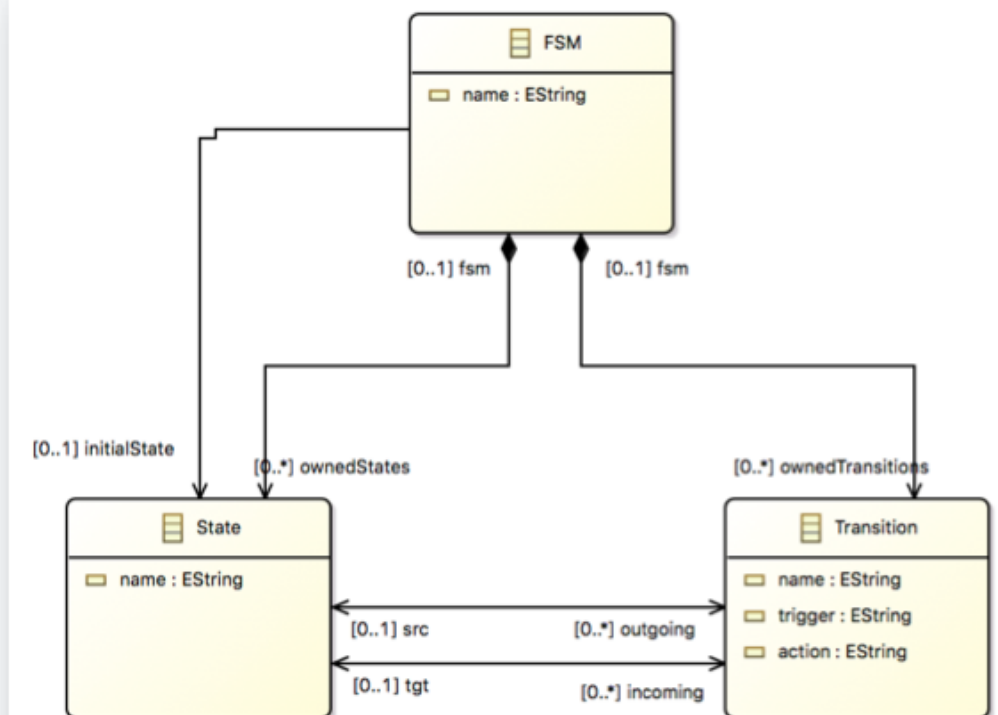
@Aspect(className=State)
class StateAspect {
    @Step
    def public void step(String inputString) {
        // Get the valid transitions
        val validTransitions = _self.outgoing.filter[t | inputString.compareTo(t.trigger) == 0]

        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

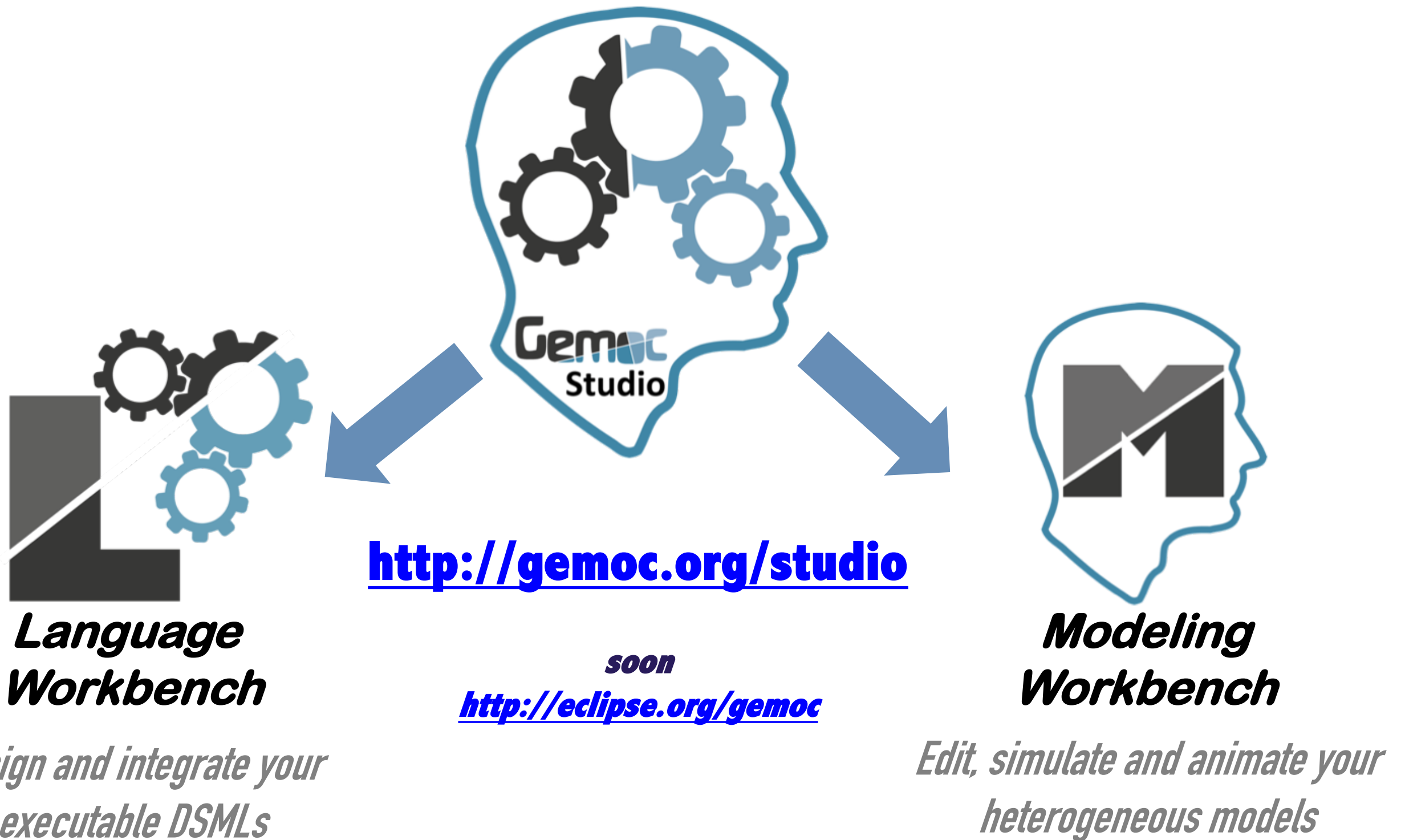
}

```

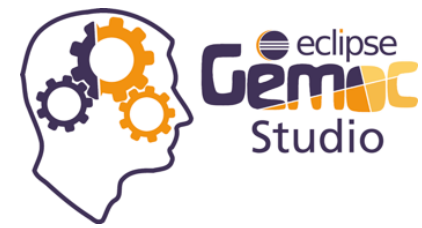
Labs: Build a VM for your StateMachine DSL

- ▶ Implement the semantics with Xtend (/Kermeta)
- ▶ [optionel] Implement (or adapt from the MODELS tutorial) the graphical representation (Editor/Animator)
- ▶ Use GEMOC to get for free your modeling environment with model execution, animation and debugging
- ▶ <https://github.com/gemoc/MODELS2017Tutorial>

The GEMOC Studio



Arduino Designer



The screenshot shows the Eclipse Gemoc Studio interface with the Arduino Designer plugin. The main window is titled "runtime-arduinoDebug - platform:/resource/org.gemoc.sample.arduino.sequential.blinker/model Laird/Sketch - Gemoc Studio".

Debug Console: Shows the execution of the "blinker [ALE Launcher]" project. The output includes the command `(VariableDeclaration) org.gemoc.sequential.model.arduino.impl.VariableDeclarationImpl@9b02cf4 -> execute()` and the global context "Project".

Hardware View: Displays an "Arduino Board" with three digital pins (0, 1, 2) connected to three LEDs: RED LED, BLUE LED, and WHITE LED.

Sketch Editor: Shows a "newSketch" with a "Repeat 5" loop. The loop contains three LED control blocks: "BLUE LED : ((i/2)%2)", "RED LED : ((i/2)%2)", and "WHITE LED : ((i/4)%2)". Each block is connected to a digital pin (0, 1, 2) and an "int 2" variable. Below the loop is a "Set i = (i+1)" block, which is connected to a variable "i" and an "int 1" variable.

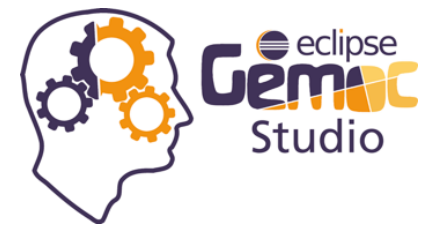
Console: Displays the "Modeling workbench console" output: "About to initialize and run the GEMOC Execution Engine..." and "Initialization done, starting engine..."

Variables Panel: Lists the variables and their values:

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>

Activity Diagram Debugger



Debug - platform:/resource/org.modelexecution.operationalsemantics.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 eXecutable Model]

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

(x) Variables

Name	Value
heldTokens (ActivityFinalNode_finalNode2 :ActivityFinalNode)	0
heldTokens (ForkNode_forkNode1 :ForkNode)	0
heldTokens (InitialNode_initialNode2 :InitialNode)	[activitydiagram.impl.ControlTokenImpl]
heldTokens (JoinNode_joinNode1 :JoinNode)	0

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>