MODEL EXECUTION

BUILD YOUR OWN COMPILER AND VIRTUAL MACHINE

MASTER 1 ICE, 2020-2021

BENOIT COMBEMALE
PROFESSOR, UNIV. RENNES 1 & INRIA, FRANCE

HTTP://COMBEMALE.FR
BENOIT.COMBEMALE@IRISA.FR
@BCOMBEMALE
Reminder about what is a language

- **Abstract Syntax (AS)**
- **Concrete Syntax (CS)**
- **Semantics Domain (SD)**

Relations:
- $\text{Mac}$ from **Concrete Syntax (CS)** to **Abstract Syntax (AS)**
- $\text{Mas}$ from **Abstract Syntax (AS)** to **Semantics Domain (SD)**
Reminder about what is an abstract syntax
Reminder about what is a concrete syntax

Domain Model (abstract syntax) → Configuration Model (mapping) → Textual or Graphical Representation (concrete syntax)

Compilation or interpretation → textual or graphical editor
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

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

- **Axiomatic**
  ```java
class System {
    x : Int

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

- **Denotational/translational**

- **Operational**
  ```java
  operation decr () is do
    if x>0 then x = x - 1 end
  ```
Definition of the Behavioral Semantics of DSL

Benoit Combemale, Feb. 2021
Implement your own compiler / interpreter

- Visitor-based?
  - Interpreter/visitor patterns, static introduction (aka. open class)
Implement your own compiler with Xtend/K3

Benoit Combemale, Feb. 2021
Implement your own interpreter with Xtend/K3

```java
@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.isEmpty) {
            // 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.inProcessTrigger
    }
}

@Aspect(className=FSM)
class FSMAspect {
    public State currentState
}
```
Part 3: define a compiler from your language to Java
Part 4 (optional): define an interpreter for your language.
The GEMOC Studio

Design and integrate your executable DSMLs

http://gemoc.org/studio

and

http://eclipse.org/gemoc

Edit, simulate and animate your heterogeneous models

Model Execution (M1ICE)
Benoit Combemale, Feb. 2021
Arduino Designer

https://github.com/gemoc/arduino-modeling

Model Execution (M1ICE)
Benoit Combemale, Feb. 2021
UML Activity Diagram

https://github.com/gemoc/activitydiagram
xCapella