Implementing Domain-Specific Languages

pictures: sxc.hu

UNIVERSITÉ CÔTE D'AZUR

Sébastien Mosser (UCA, I3S)

ENS Lyon, 15.09.2017

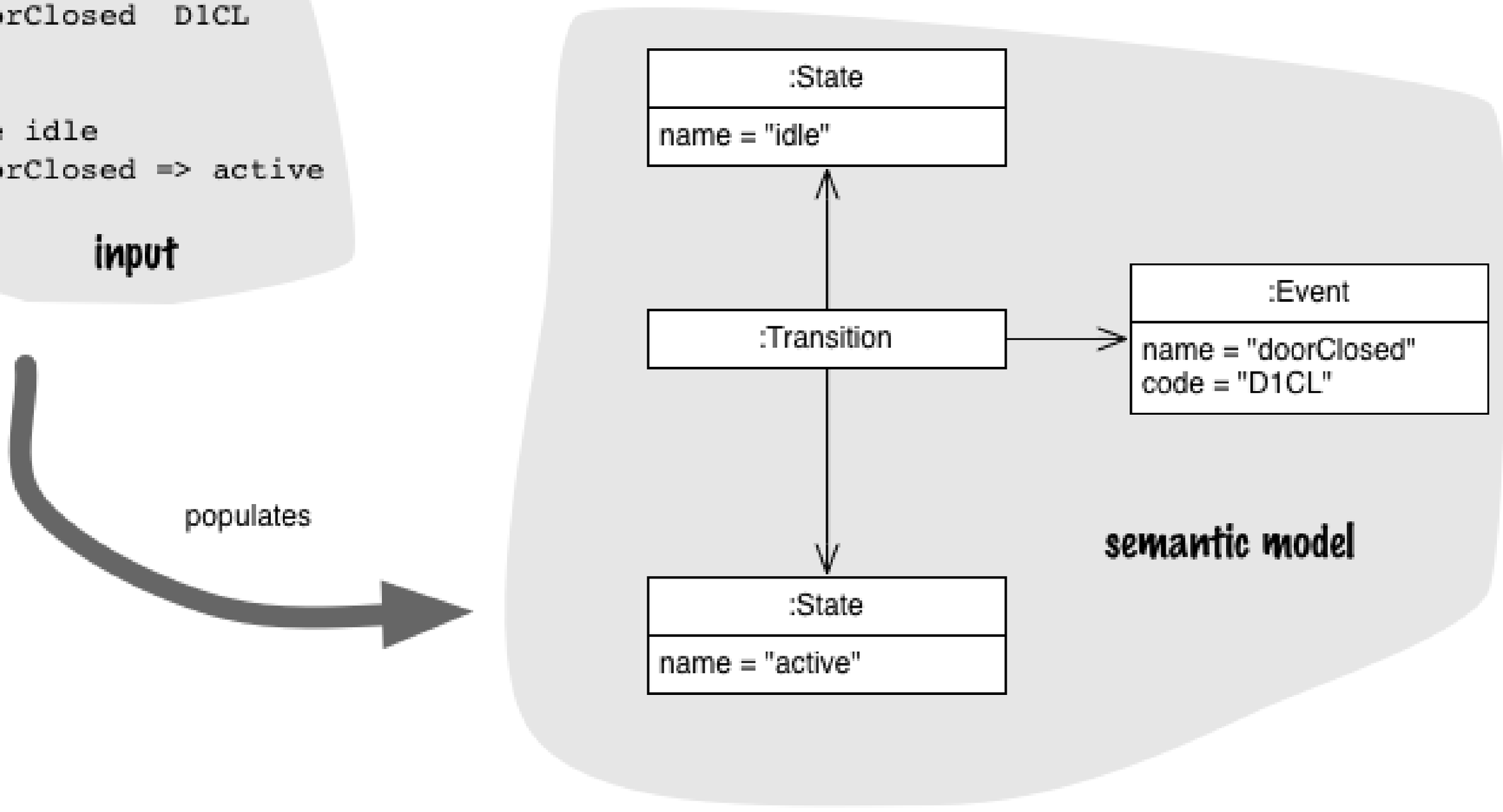
External DSLs

```
events
  doorClosed     D1CL
  drawerOpened   D2OP
  lightOn        L1ON
  doorOpened     D1OP
  panelClosed    PNCL
end

resetEvents
  doorOpened
end

commands
  unlockPanel PNUL
  lockPanel   PNLK
  lockDoor    D1LK
  unlockDoor  D1UL
end
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end

state active
  drawerOpened => waitingForLight
  lightOn => waitingForDrawer
end

state waitingForLight
  lightOn => unlockedPanel
end

state waitingForDrawer
  drawerOpened => unlockedPanel
end

state unlockedPanel
  actions {unlockPanel lockDoor}
  panelClosed => idle
end
```

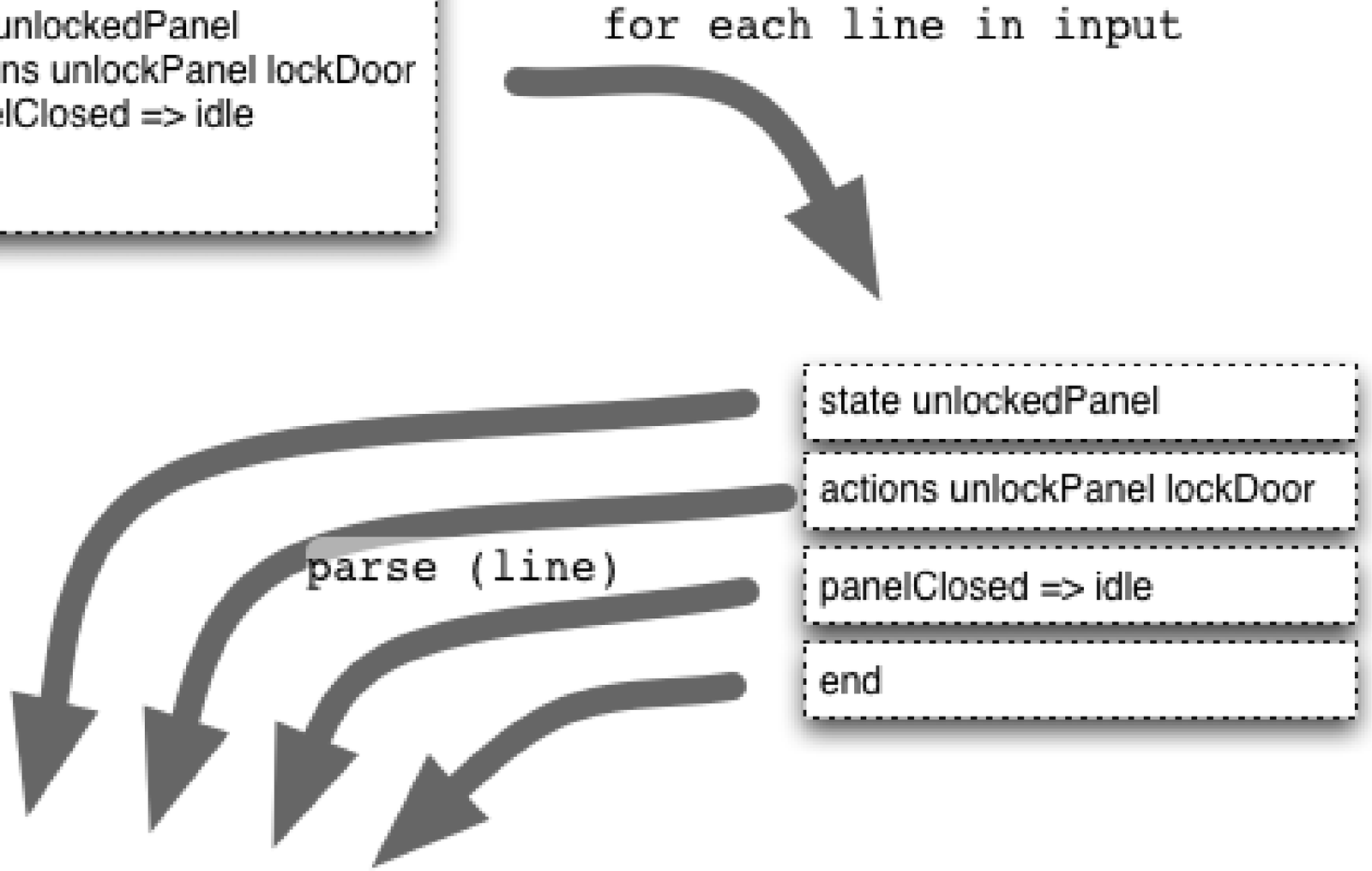
```
events
  doorClosed  D1CL
end

state idle
  doorClosed => active
end
```

input

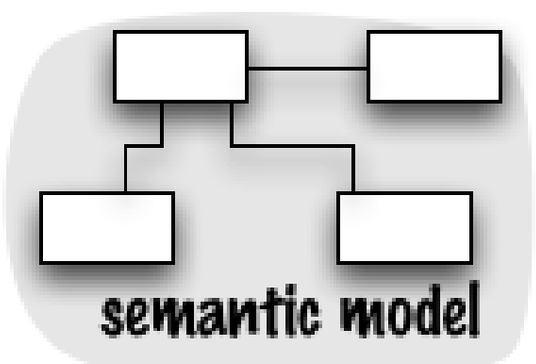


Delimiter-directed Translation

```
state unlockedPanel
  actions unlockPanel lockDoor
  panelClosed => idle
end
```

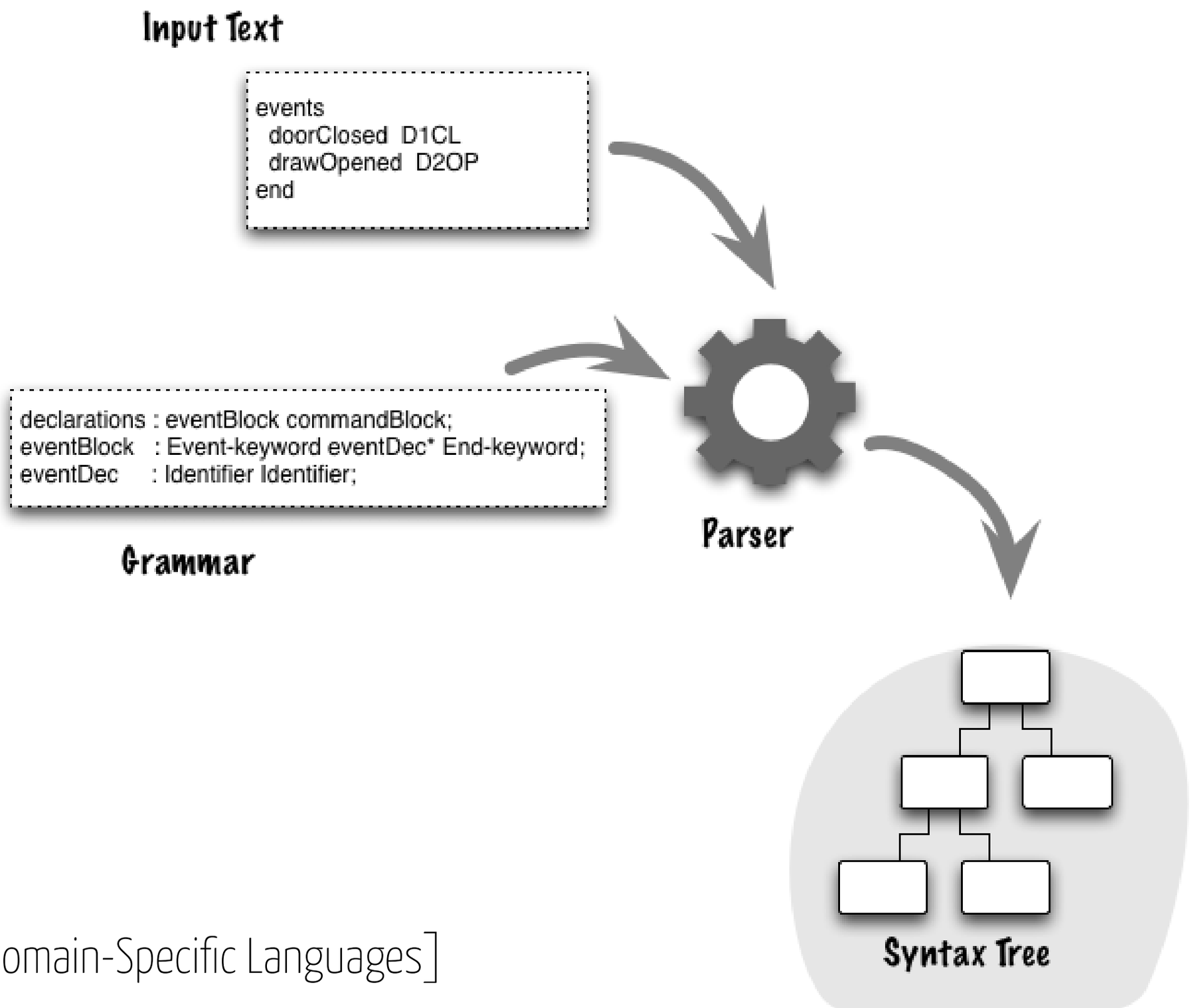


Easy



Limited

Syntax-directed Translation



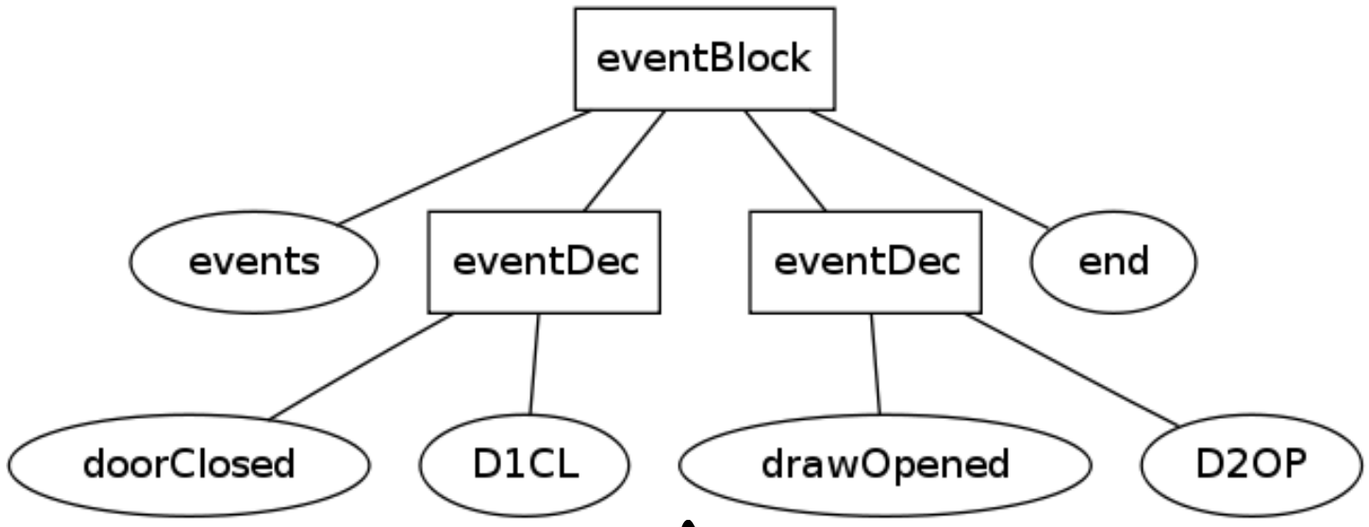
Introducing Grammars

```
declarations : eventBlock commandBlock;

eventBlock : Event-keyword eventDec* End-keyword;

eventDec : Identifier Identifier;

commandBlock : Command-keyword commandDec* End-keyword;

commandDec : Identifier Identifier;
```

This is not a compilation course!

From **Words** to **Parse Tree**

```
events
   doorClosed  D1CL
   drawOpened  D2OP
end
```



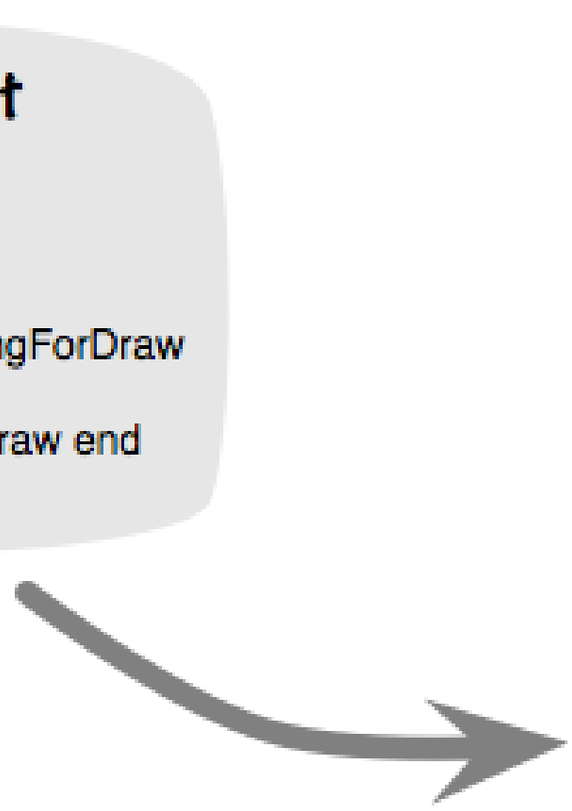
lexer

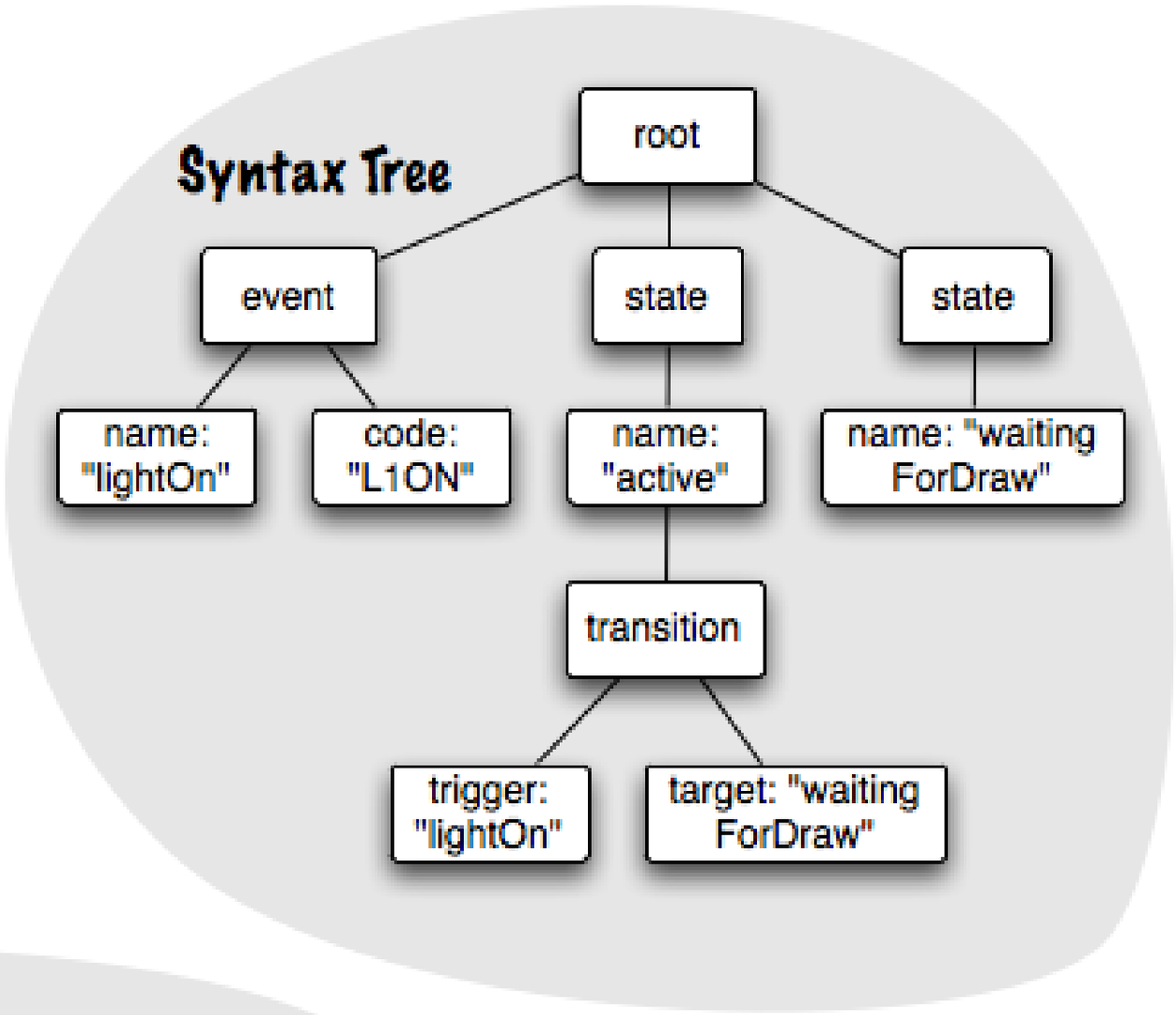
parser

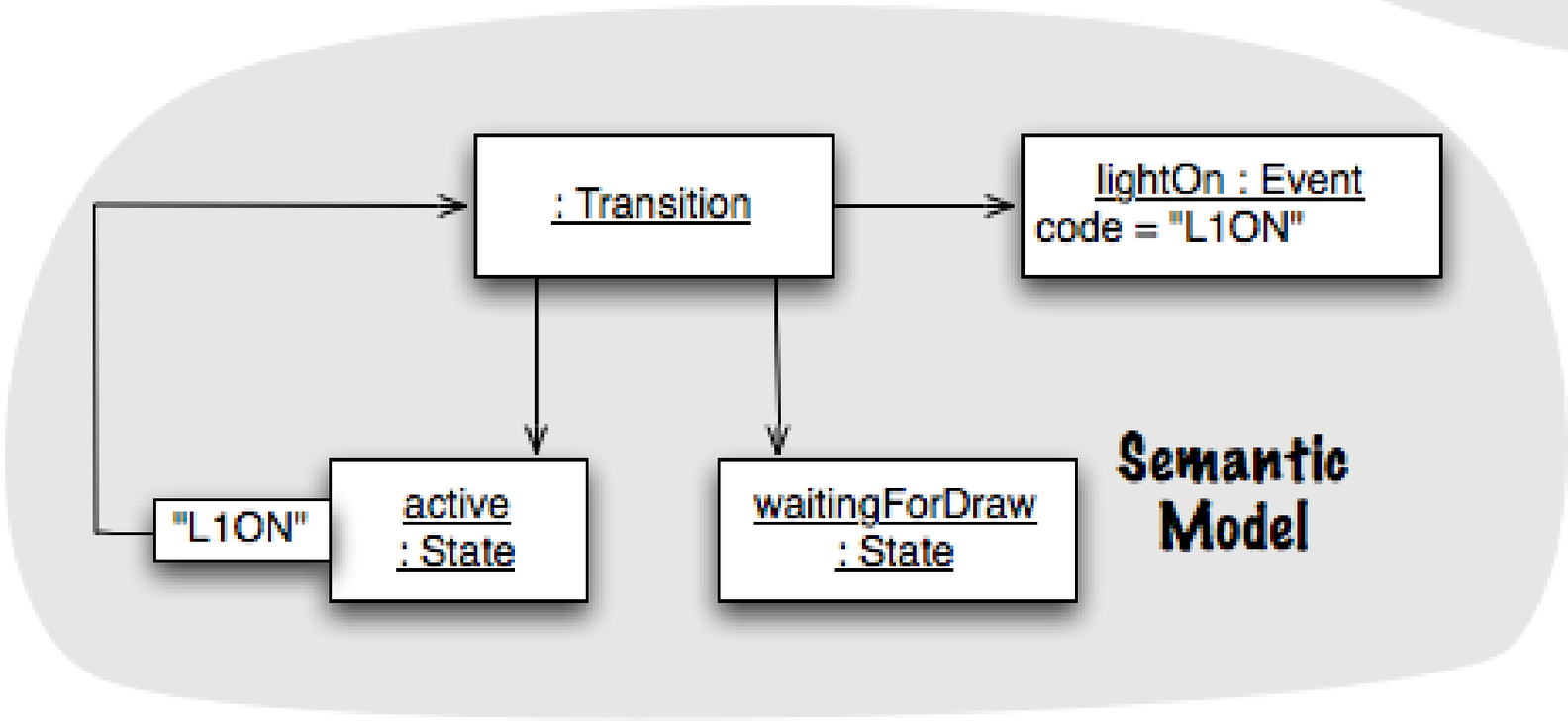
```
[Event-keyword: "events"]
[Identifier: "doorClosed"]
[Identifier: "D1CL"]
[Identifier: "drawOpened"]
[Identifier:"D2OP"]
[End-keyword: "end"]
```

DSL Text

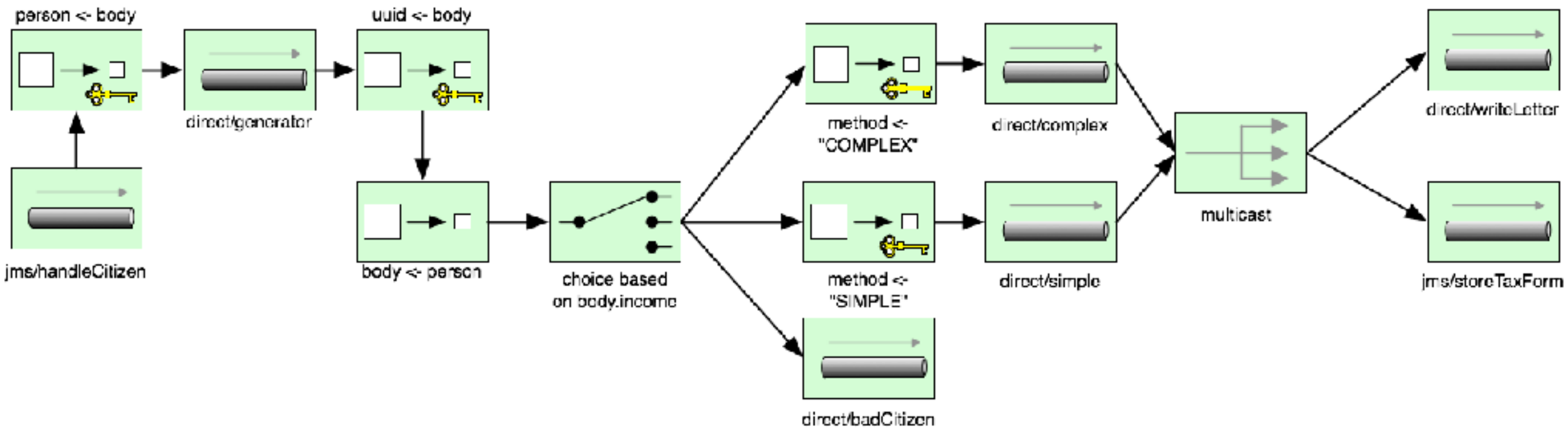
```
events
  lightOn L1ON
end
state active
  lightOn => waitingForDraw
end
state waitingForDraw end
```







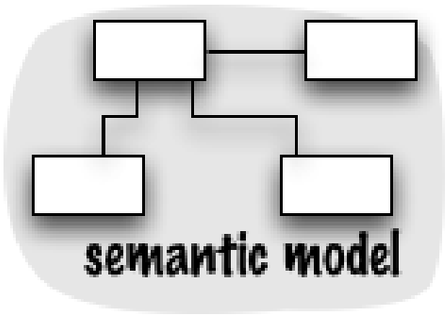
Graphical DSL



e.g., enterprise integration patterns

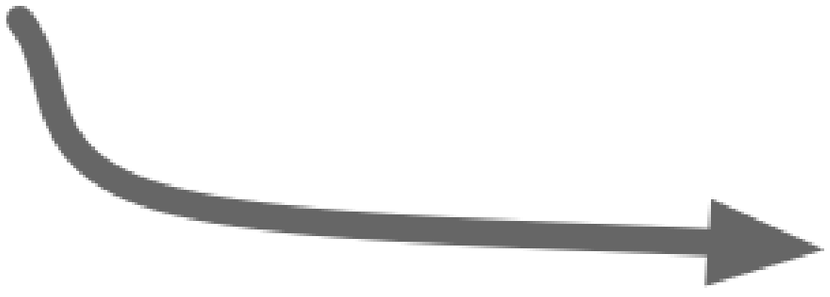
Code Generation

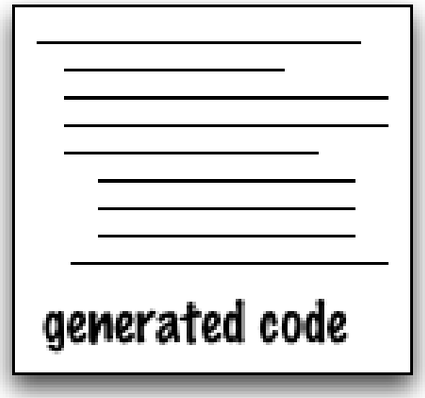
Transformer Generation



ToArduinoCode.java

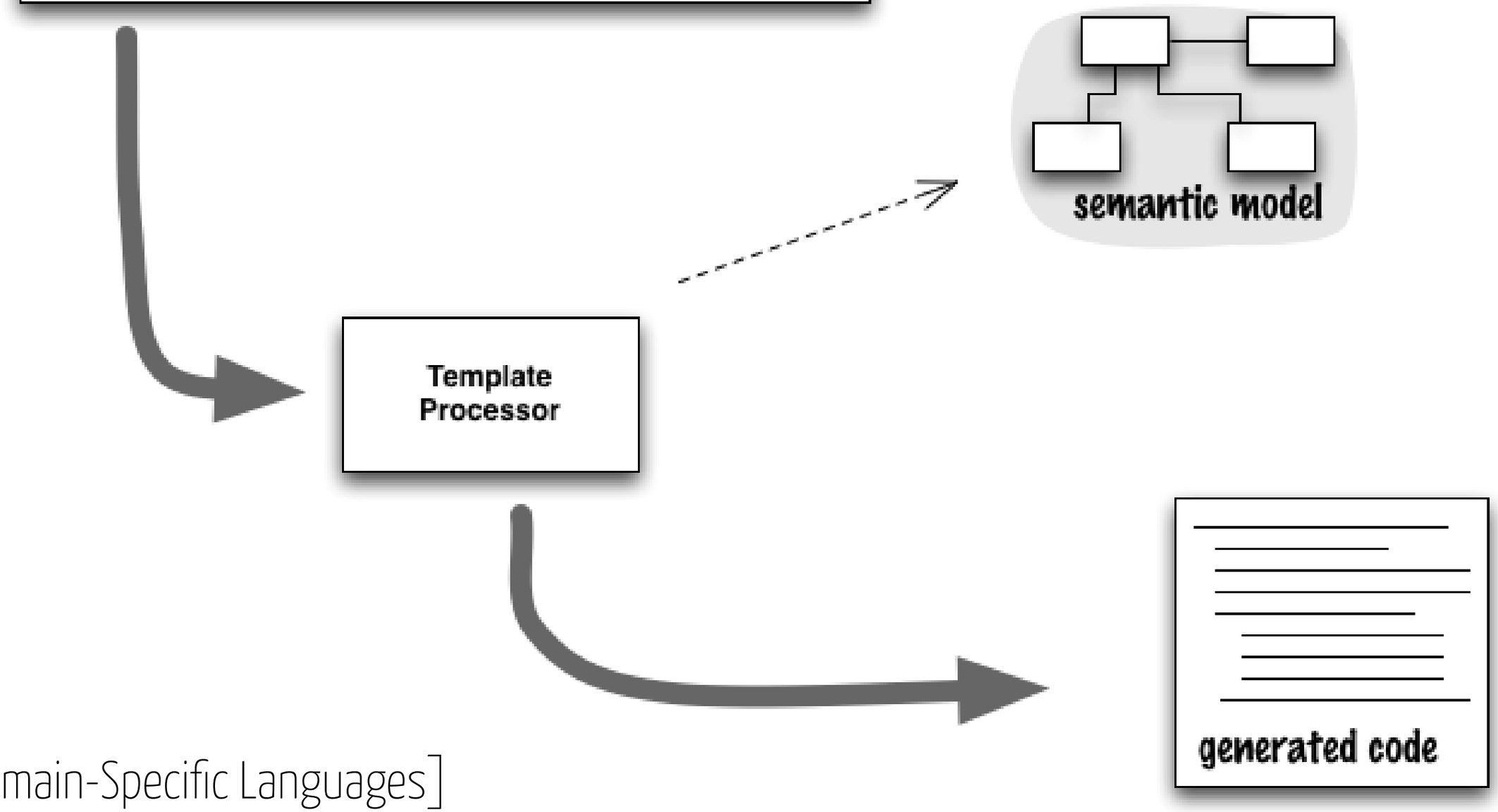
```
private void generateEvents(Writer output) throws IOException {
  for (Event e : machine.getEvents())
    output.write(String.format("  declare_event(\"%s\", \"%s\");\n",
                               e.getName(), e.getCode()));
  output.write("\n");
}
```



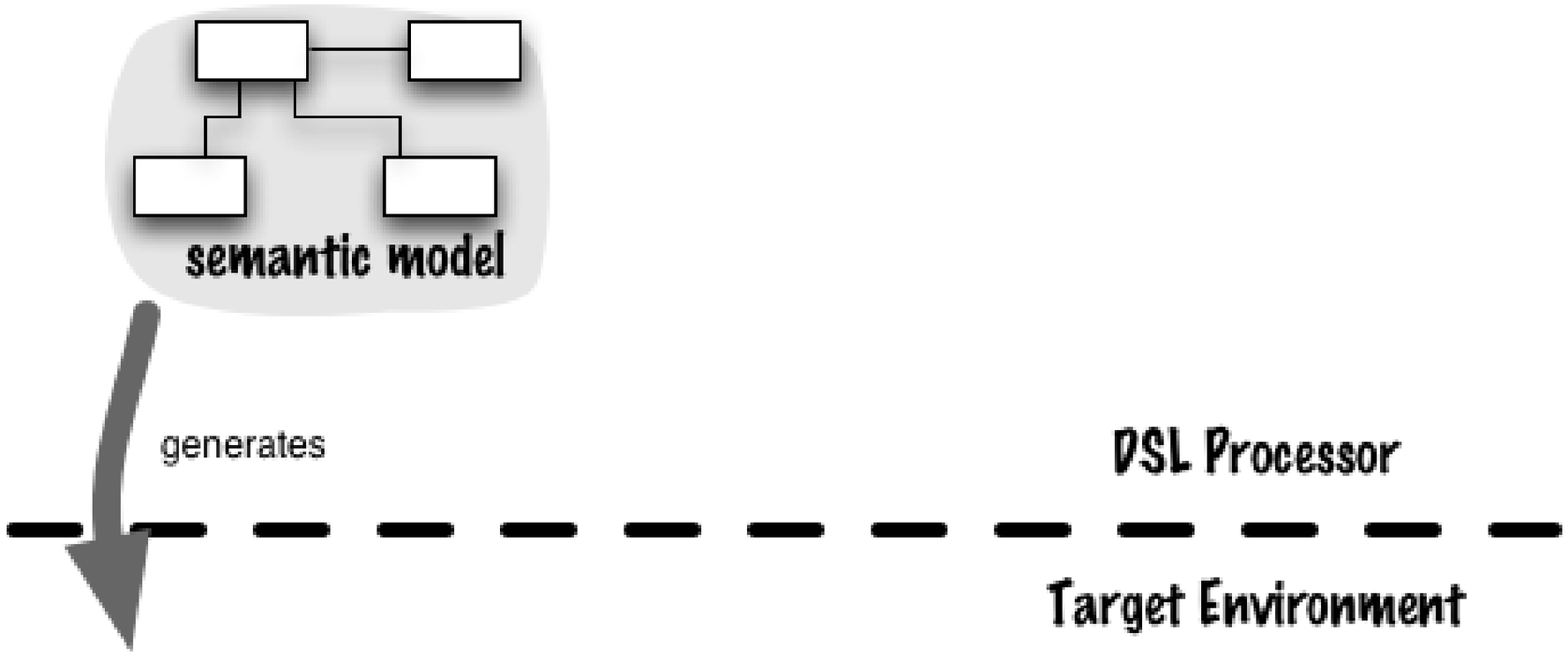


Template-based generation

```
#foreach($e in $helper.events)
  declare_event("${e.name}", "${e.code}");
#end
```

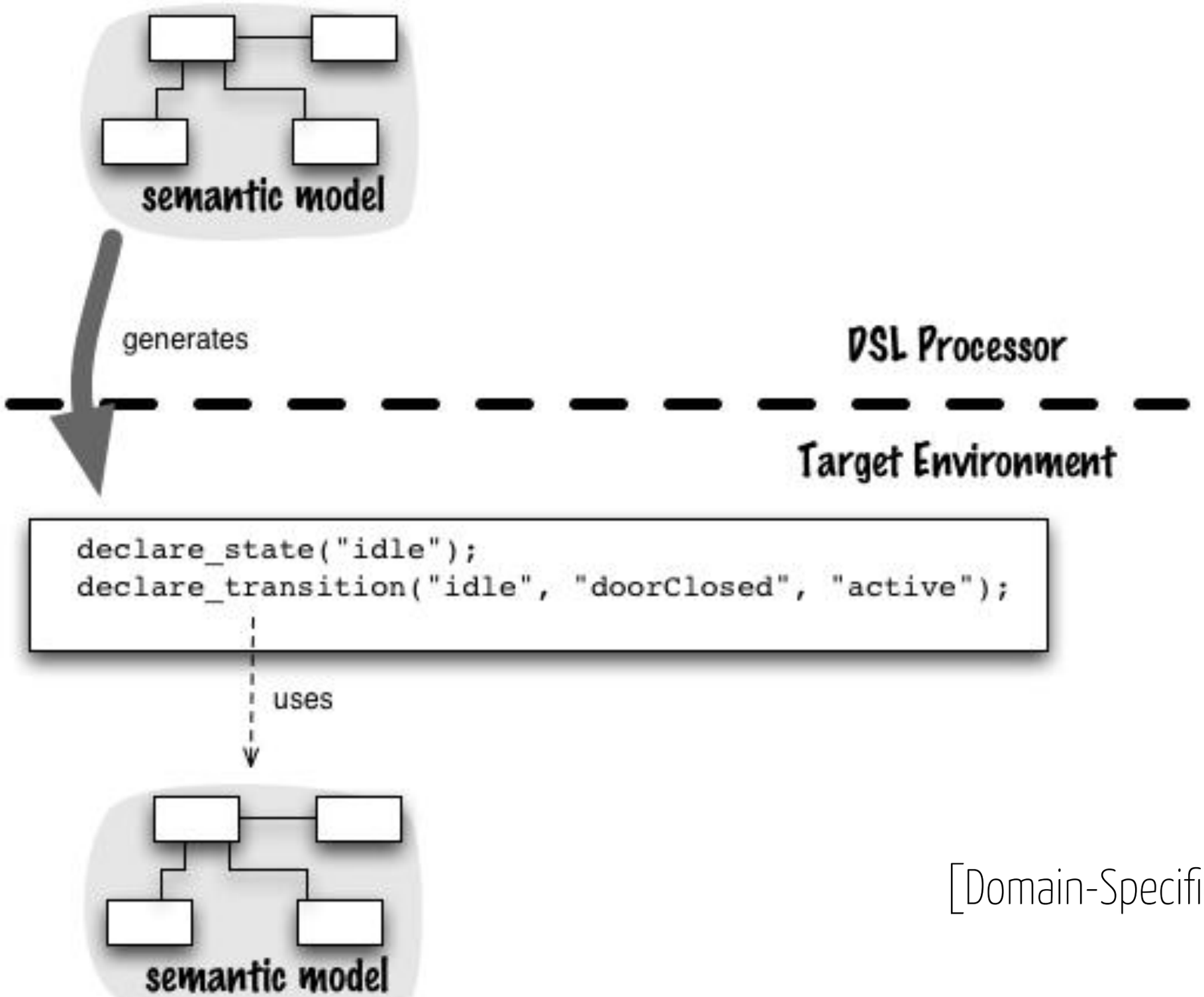


Model-ignorant Generation




```
void handle_event(char *code) {
  switch(current_state_id) {
  case STATE_idle: {
    if (0 == strcmp(code, EVENT_doorClosed)) {
      current_state_id = STATE_active;
    }
    return;
  }
  case STATE_active: {
 ...
```

Model-aware Generation



Embedded DSLs

```haskell
main :: IO ()
main = either print print . fmap generate $ buildApp "example" $ do
  addSensor button $ onPin 9
  addActuator light $ onPin 12
  defineStates [offline, online]
  actionsWhen offline `execute` [ set light `to` off ]
  actionsWhen online  `execute` [ set light `to` on ]
  transitionsFrom online  `are` [ when button `is` pressed $ goto offline ]
  transitionsFrom offline `are` [ when button `is` pressed $ goto online ]
  start offline
```

Modeling Controllers

- **Events**:
  - A code sent **by** the environment
  - Inform about context changes
- **Commands**:
  - A code sent **to** the environment
  - Change the context (e.g., unlock)
- **States**
  - Controller's current situation
  - Might trigger Commands
  - Reacts to events with Transitions
- **Transitions**
  - Associate an event to a next State

```
events
  doorClosed     D1CL
  drawerOpened   D2OP
  lightOn        L1ON
  doorOpened     D1OP
  panelClosed    PNCL
end

resetEvents
  doorOpened
end

commands
  unlockPanel PNUL
  lockPanel   PNLK
  lockDoor    D1LK
  unlockDoor  D1UL
end
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end

state active
  drawerOpened => waitingForLight
  lightOn => waitingForDrawer
end

state waitingForLight
  lightOn => unlockedPanel
end

state waitingForDrawer
  drawerOpened => unlockedPanel
end

state unlockedPanel
  actions {unlockPanel lockDoor}
  panelClosed => idle
end
```

```
events
  doorClosed    D1CL
end
```

```
commands
  lockPanel   PNLK
  unlockDoor  D1UL
end
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end
```

```
 state active
   ...
 end
```

```
events
  doorClosed    D1CL
end
```

```
event :doorClosed, "D1CL"
```

```
commands
  lockPanel   PNLK
  unlockDoor  D1UL
end
```

```
command :lockPanel, "PNLK"
command :unlockDoor, "D1UL"
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end
```

```
state :idle do
  actions :unlockDoor, :lockPanel
  transitions :doorClosed => :active
end
```

```
 state active
   ...
 end
```

```
state :active do
  ...
end
```

```
event :doorClosed, "D1CL"
```

```
command :lockPanel, "PNLK"
command :unlockDoor, "D1UL"
```

```
state :idle do
  actions :unlockDoor, :lockPanel
  transitions :doorClosed => :active
end
```

```
state :active do
  ...
end
```

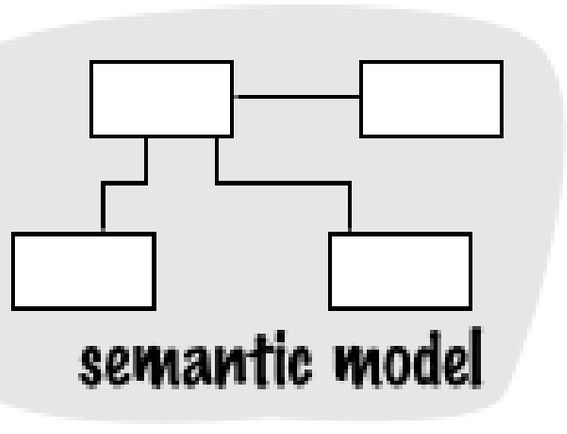
input text

```
computer()
  .processor()
    .cores(2)
    .i386()
```

Computer Builder

processor()
cores(int)
i386()

```
processor = new Processor(2, Processor.Type.i386)
```



semantic model

The **Calendar** Example

```
class Calendar {
  private List<Event> events = new ArrayList<Event>();
  // ...
}

class Event {
  private String name, location;
  private LocalDate date;
  private LocalTime startTime, endTime;
  // ...
}


Calendar c = new Calendar();
Event e1 = new Event(«DSL Tutorial»);
e1.setLocation(...)
...
c.add(e1)
```

Designing **Fluent Interfaces**

```
CalendarBuilder builder = new CalendarBuilder();

builder
   .add("DSL tutorial")
      .on   (2009, 11, 8)
      .from("09:00")
      .to   ("16:00")
      .at   ("Aarhus Music Hall")
   .add("Making use of Patterns")
      .on   (2009, 10, 5)
      .from("14:15")
      .to   ("15:45")
      .at   ("Aarhus Music Hall")
;

calendar = builder.getContent();
```

Designing **Fluent Interfaces**

```
CalendarBuilder builder = new CalendarBuilder();

builder
   .add("DSL tutorial")
      .on   (2009, 11, 8)
      .from("09:00")
      .to   ("16:00")
      .at   ("Aarhus Music Hall")
   .add("Making use of Patterns")
      .on   (2009, 10, 5)
      .from("14:15")
      .to   ("15:45")
      .at   ("Aarhus Music Hall")
;

calendar = builder.getContent();
```

Ugly Java

Good Calendar

Method Chaining

```
class CalendarBuilder { // Excerpt

  private List<EventBuilder> events =
    new ArrayList<EventBuilder>();

  public EventBuilder add(String name) {
    EventBuilder child = new EventBuilder(this);
    events.add(child); child.setName(name);
    return child;
  }
}

class EventBuilder {  // Excerpt
  private String Location;

  public EventBuilder at (String location) {
    this.location = location;
    return this;
  }
}
```

```
builder
  .add("DSL tutorial")
    .at ("Aarhus Music Hall")
```

Function Sequence

```
computer();
  processor();
    cores(2);
    speed(2500);
    i386();
disk();
  size(150);
disk();
  size(75);
  speed(7200);
  sata();
```

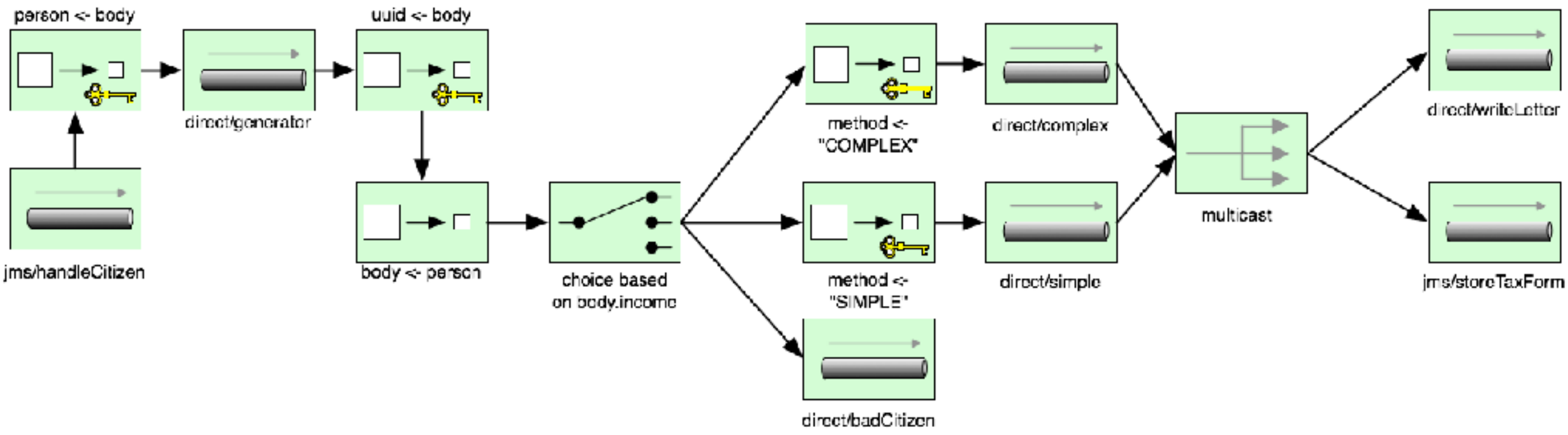
```
class ComputerBuilder {

  void computer() {
    currentDisk = null;
    currentProcessor = null;
  }

  void processor() {
    currentProcessor =
        new ProcessorBuilder();
    processor = currentProcessor;
    currentDisk = null;
  }

  void cores(int arg) {
    currentProcessor.cores = arg;

  }
```

Nested Functions

```
computer(
  processor(
    cores(2),
    speed(2500),
    i386),
  disk(
    size(150)
  ),
  disk(
    size(75),
    speed(7200),
    SATA
  )
);
```

```
class Builder {

  static Computer computer(Processor p,
                           Disk... d) {
    return new Computer(p, d);
  }

  static Processor processor(
    int cores, int speed, Type type) {
    return new Processor(cores, speed, type);
  }

  static int cores(int value) {
    return value;
  }

  static final Type i386 = Type.i386;

}
```

Graphical DSL

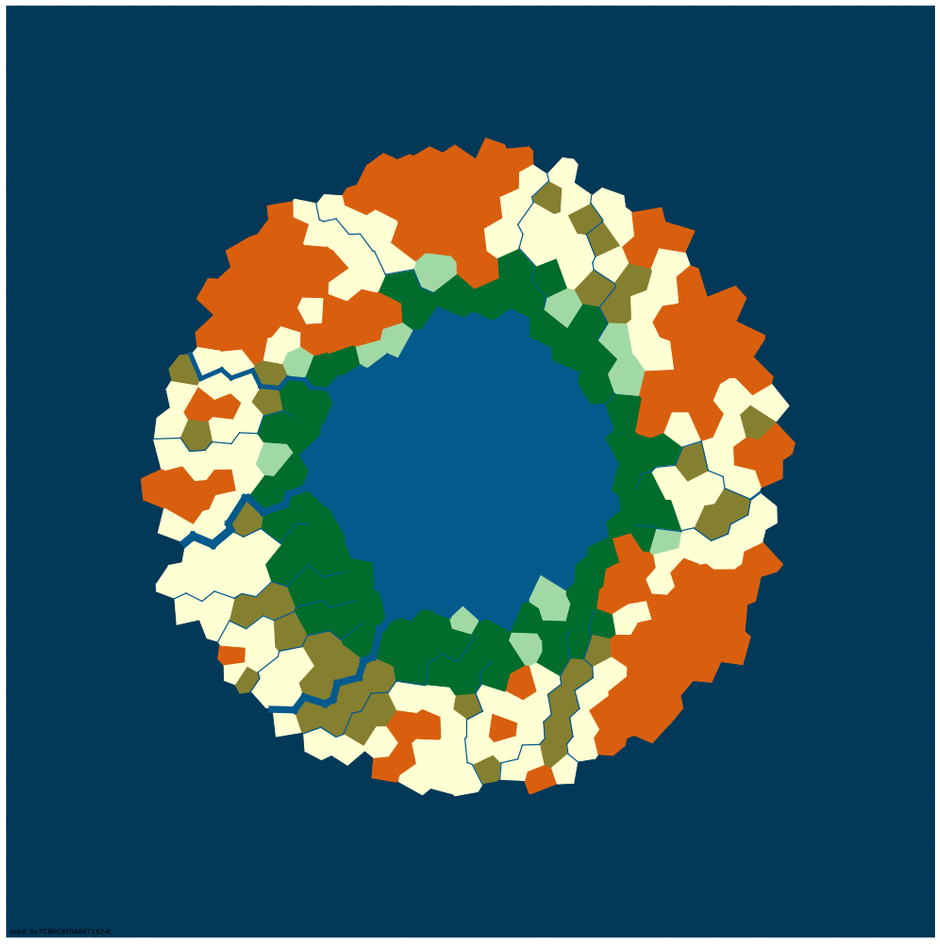


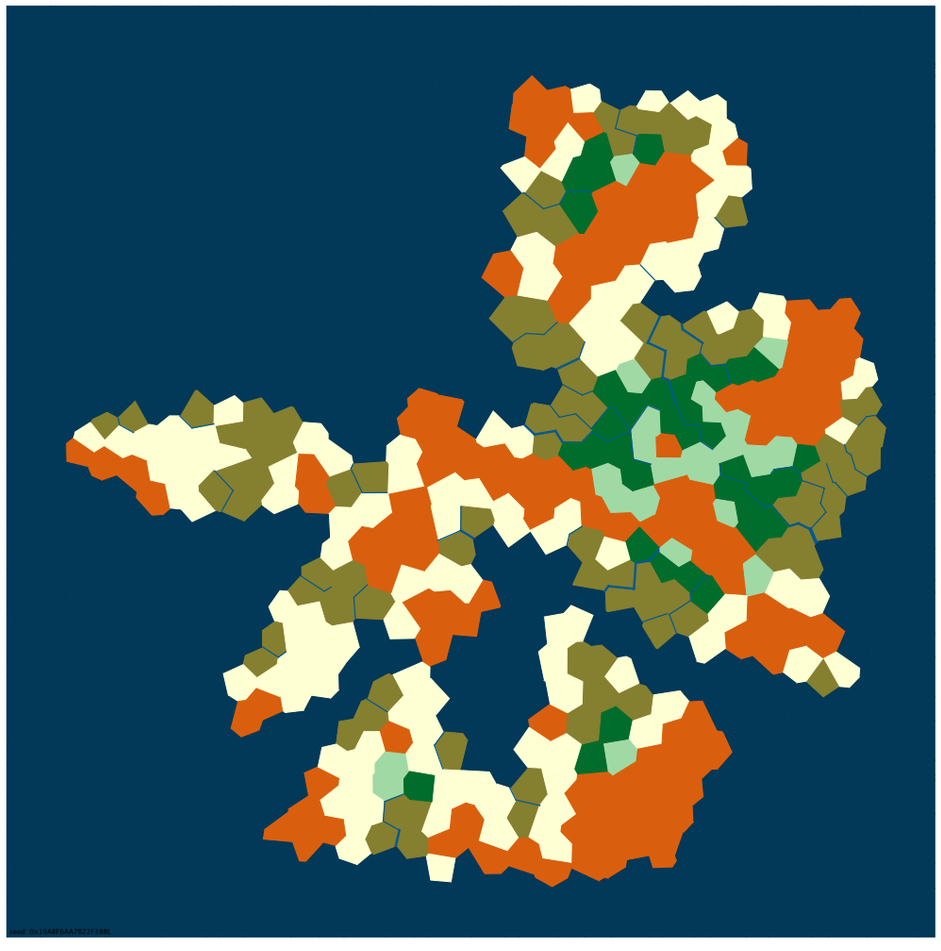
e.g., enterprise integration patterns

```
// Route to handle a given Person
from(HANDLE_CITIZEN)
                .log("    Routing ${body.lastName} according to income ${body.income}")
                .log("      Storing the Person as an exchange property")
                .setProperty("person", body())
                .log("      Calling an existing generator")
                .to("direct:generator")
                .setProperty("p_uuid", body())
                .setBody(simple("${property.person}"))
                .choice()
                        .when(simple("${body.income} >= 42000"))
                                .setProperty("tax_computation_method", constant("COMPLEX"))
                                .to("direct:complexTaxMethod")
                        .when(simple("${body.income} >= 0 && ${body.income} < 42000"))
                                .setProperty("tax_computation_method", constant("SIMPLE"))
                                .to("direct:simpleTaxMethod")
                        .otherwise()
                                .to("direct:badCitizen").stop() // stopping the route for bad citizens
                .end() // End of the content-based-router
                .setHeader("person_uid", simple("${property.person.uid}"))
                .multicast()
                        .parallelProcessing()
                        .to("direct:generateLetter")
                        .to(STORE_TAX_FORM)
;
```

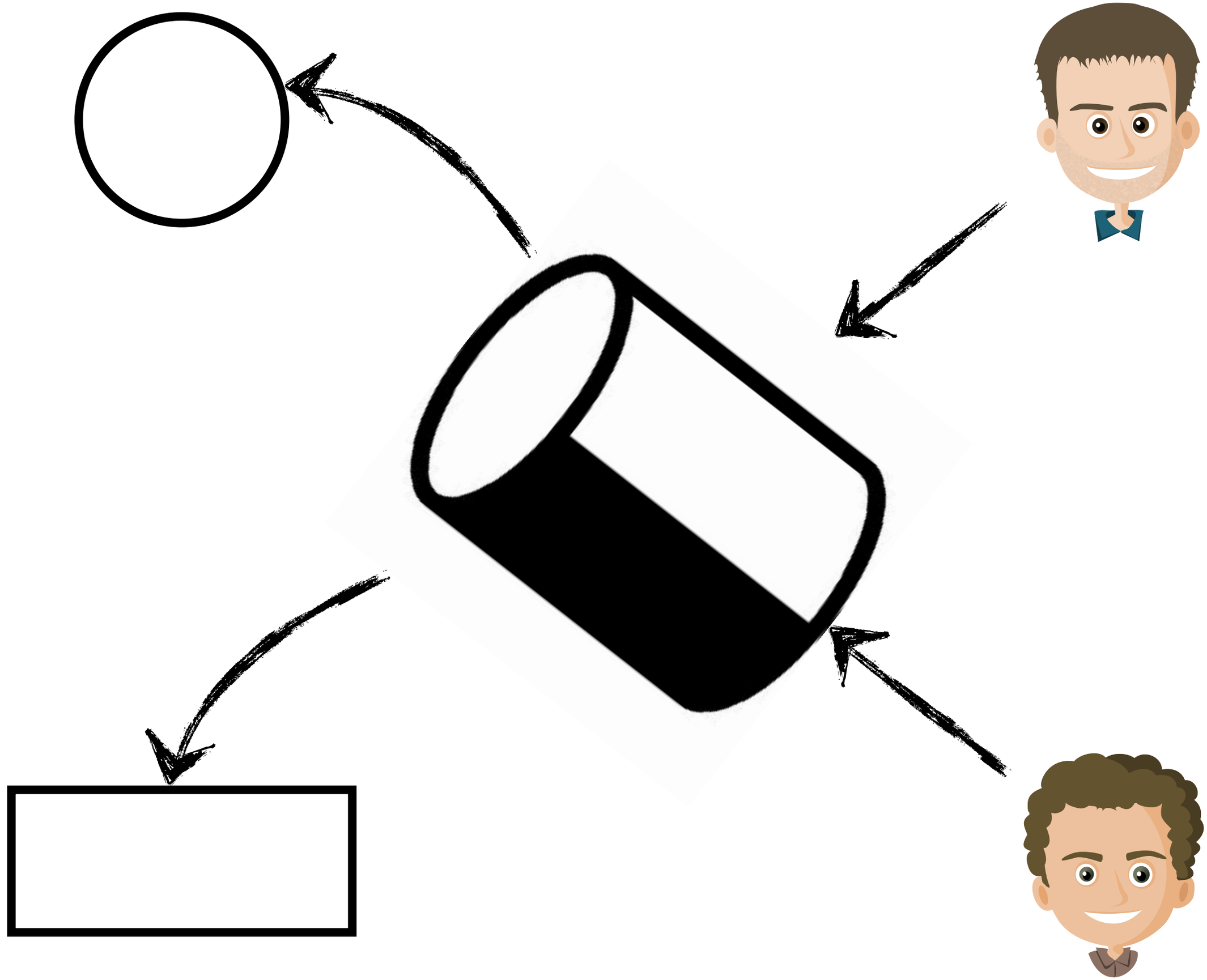
Embedded DSL

```scala
// Round island, quite big. Easy to exploit.
val s47 = 0x7C86C8F0AE471824L
lazy val week47: IslandMap = {
  createIsland shapedAs donut(70.percent, 30.percent) withSize 1600 having 1200.faces builtWith Seq(
    plateau(30), flowing(rivers = 30, distance = 0.4), withMoisture(soils.normal, distance = 700),
    AssignPitch, usingBiomes(WhittakerDiagrams.caribbean)) usingSeed s47
}
```





```scala
// Needle in an haystack
val s49 = 0x19ABF6AA7B22F38BL
lazy val week49: IslandMap = {
  createIsland shapedAs radial(factor = 1.57) withSize 1600 having 1200.faces builtWith Seq(
    plateau(30), flowing(rivers = 40, distance = 0.1), withMoisture(soils.wet, distance = 100),
    AssignPitch, usingBiomes(WhittakerDiagrams.caribbean)) usingSeed s49
}
```



```java
public static void main(String[] args) throws Exception {

    run(MyBot.class)
            .exploring(load("map.json"))    // A File containing a map as a JSON object
            .withSeed(0L)
            .startingAt(1, 1, "EAST")
            .backBefore(7000)
            .withCrew(15)
            .collecting(1000, "WOOD")
            .collecting(300,  "QUARTZ")
            .collecting(10,   "FLOWER")
            .storingInto("./outputs")       // The output directory must exists
            .fire();

}
```


```scala
object Week47 extends Run with SI3 {

  override val number: String = "47"

  override val seed: Long                  = Islands.s47
  override lazy val theIsland: IslandMap = Islands.week47

  override val crew: Int       = 15
  override val budget: Int     = 7000
  override val plane: Plane    = Plane(1,1,Directions.EAST)
  override val objectives: Set[(Resource, Int)] = Set((WOOD, 1000), (QUARTZ, 300), (FLOWER,10))

  override def players = all - "qad" - "qcb" - "qcc" - "qce" - "qcf"
```

```java
public static void main(String[] args) throws Exception {

    run(MyBot.class)
            .exploring(load("map.json"))    // A File containing a map as a JSON object
            .withSeed(0L)
            .startingAt(1, 1, "EAST")
            .backBefore(7000)
            .withCrew(15)
            .collecting(1000, "WOOD")
            .collecting(300,  "QUARTZ")
            .collecting(10,   "FLOWER")
            .storingInto("./outputs")       // The output directory must exists
            .fire();

}
```

```java
public Runner collecting(int amount, String resource) {
        Resource res = Resources.bindings().get(resource).get();
        this.contracts.put(res,amount);
        return this;
}
```

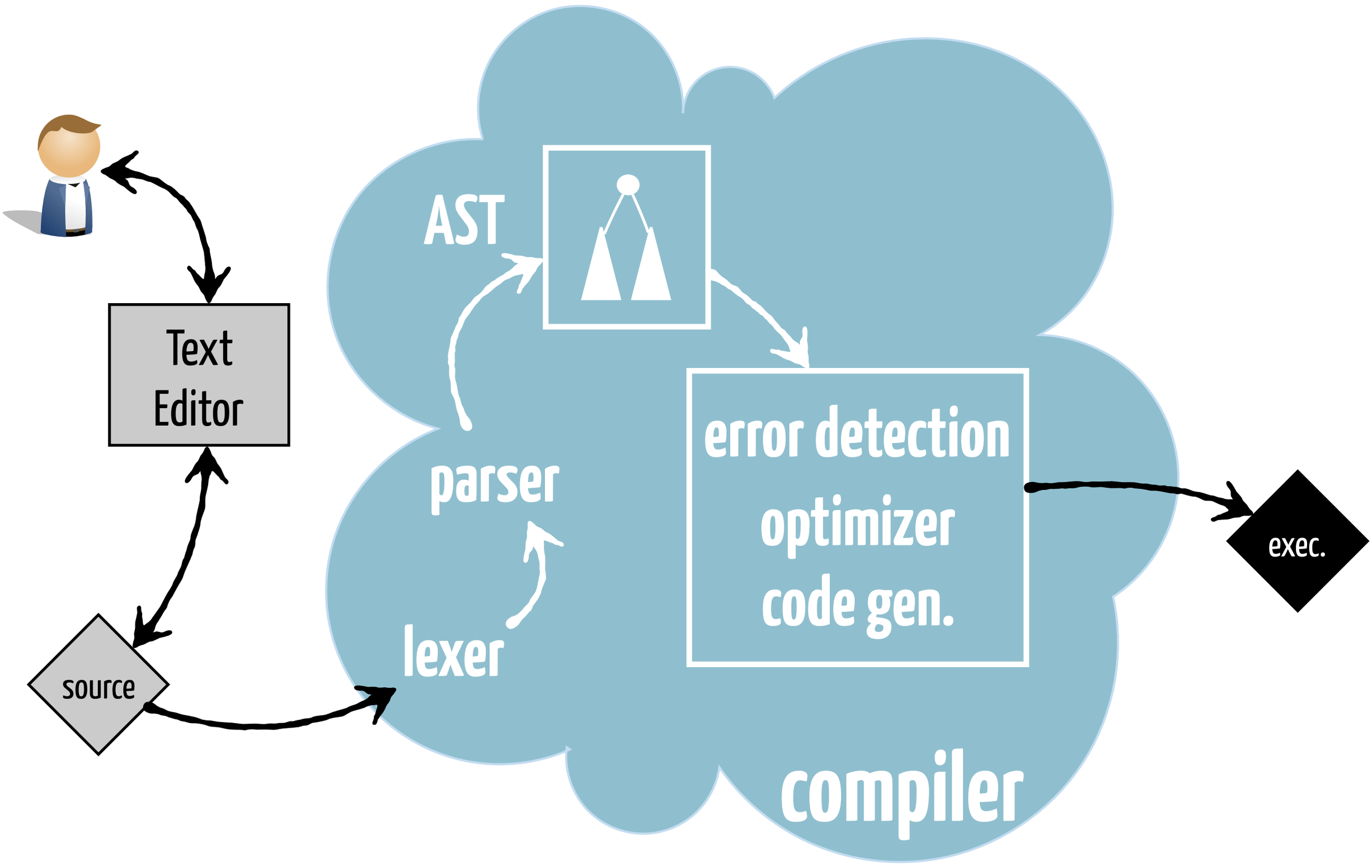
```java
public static Runner run(Class c) throws Exception {
        return new Runner(c);
}
```

```scala
// Enriching the IslandMap class to add the -> operator, used to store the map into a given file.
implicit def islandMapToEnrichedIslandMap(m: IslandMap): EnrichedIslandMap = new EnrichedIslandMap(m)
protected class EnrichedIslandMap(map: IslandMap) {
  def ->(out: IslandMap => (String, java.io.File)) {
    val result = out(map)
    result._2.renameTo(new java.io.File(result._1))
  }
}
```

Embedded implementations of DSLs can be ugly from the host language point of view

Projectional Edition

4



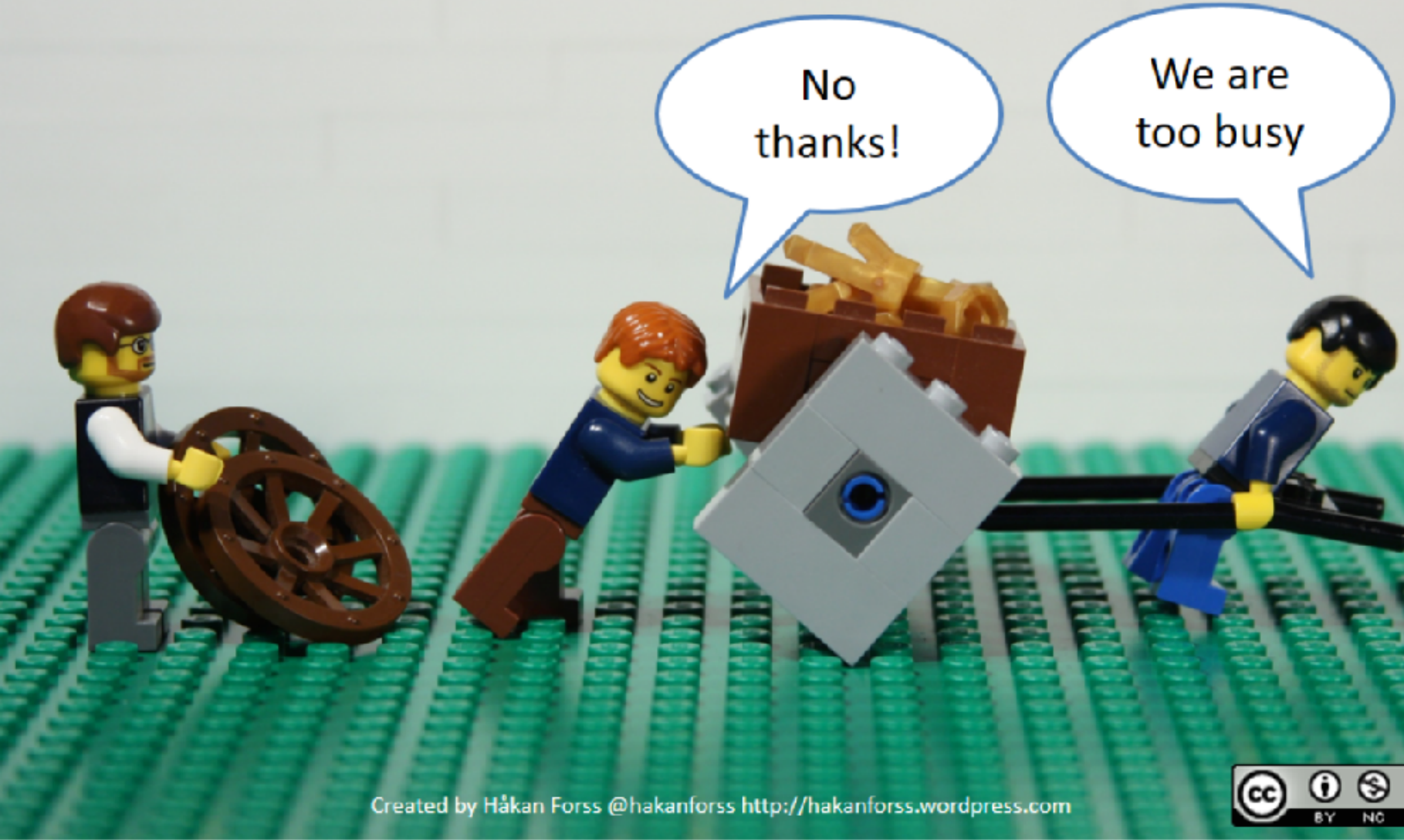
[based on Campagne's slides]

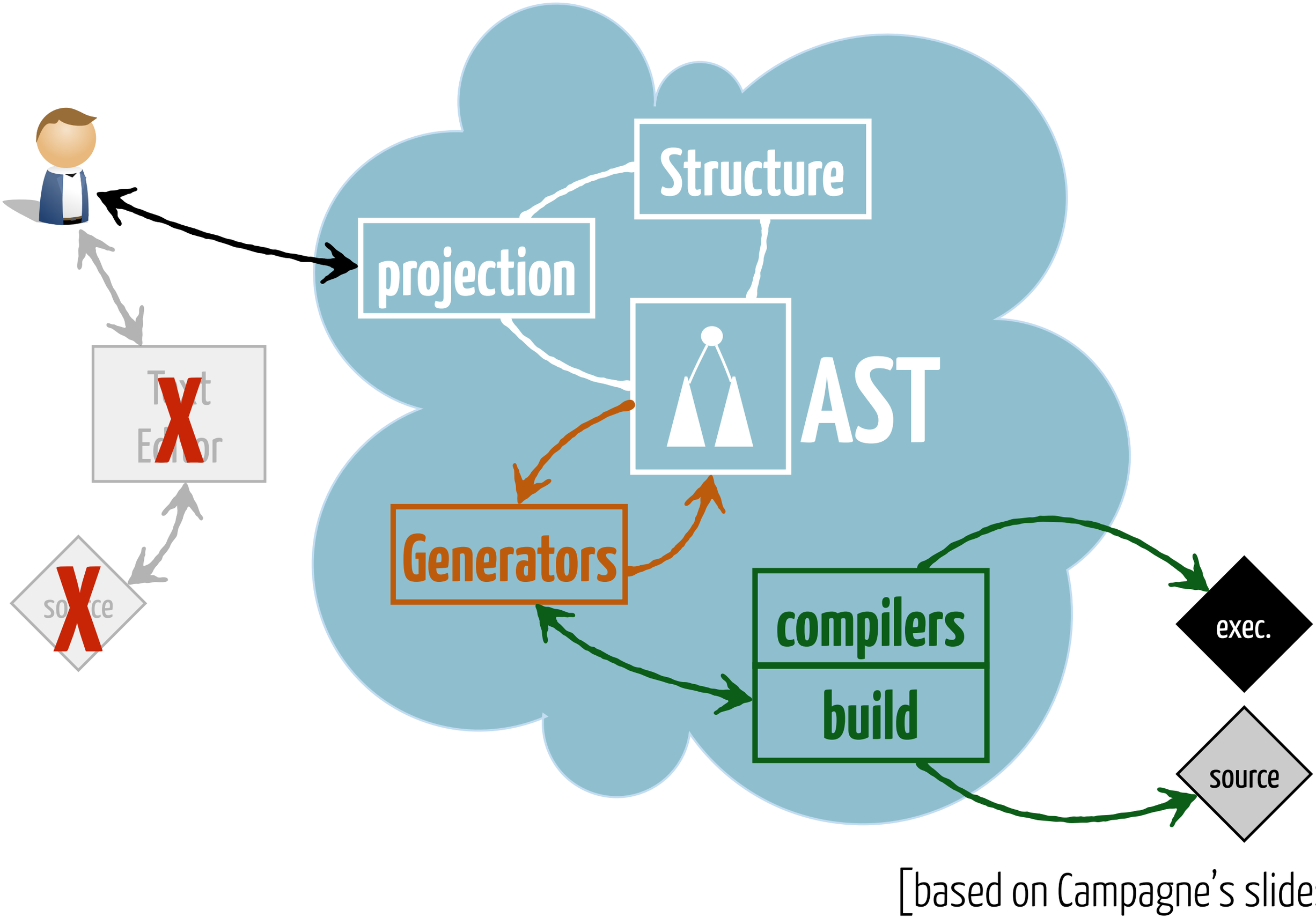
Storing **programs**

as **text**

is so **1950**

Are you too busy to improve?



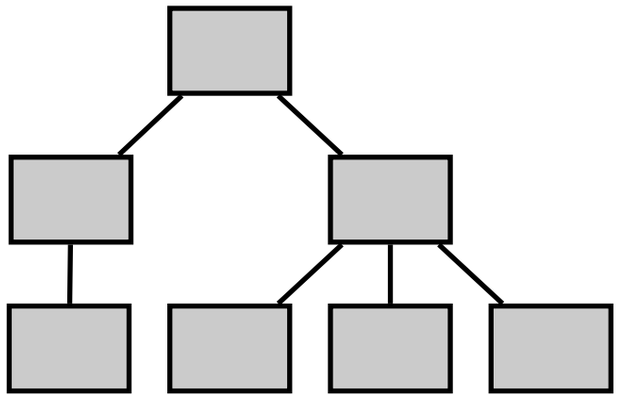


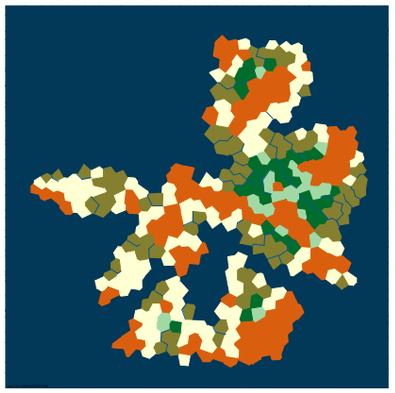
[based on Campagne's slides]

Programs are **not text** anymore!

The **AST** is the **key**

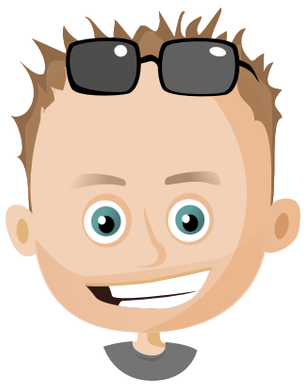
AST





P1(AST)

P2(AST)



```
Run:
  bot: MyBot

  budget: 100
  WOOD <- 400
  FLOWER <- 40

  ...
```

```
Champ [#49]
  players: 3A

  excl: "QAB"

  Contract:
    W: 100
    F: 40
```

Is projectional edition an hipster trap?

http://jeffgreenspan.com/projects/personal-work/urban-traps

Language & Environment

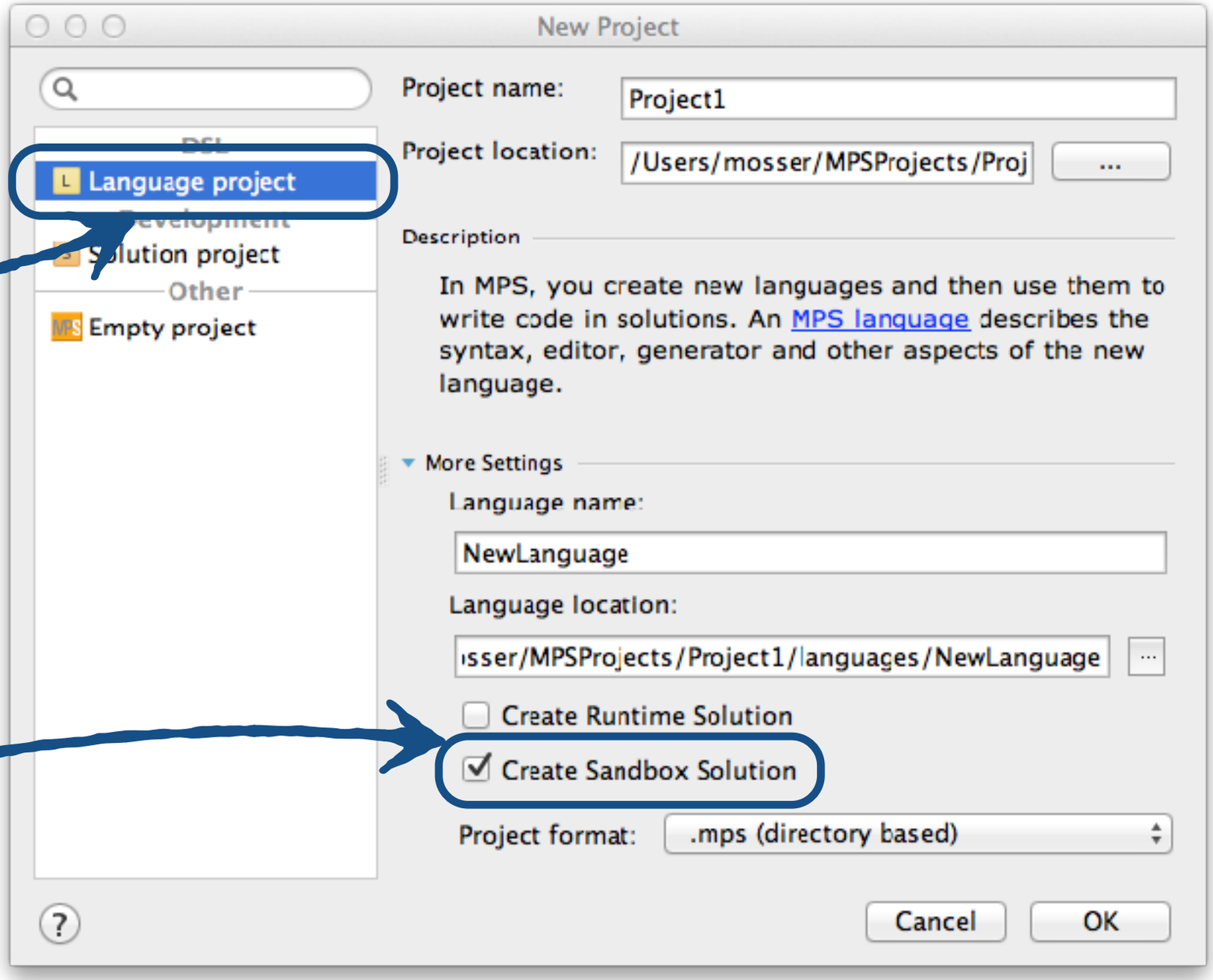
Language

(~metamodel)

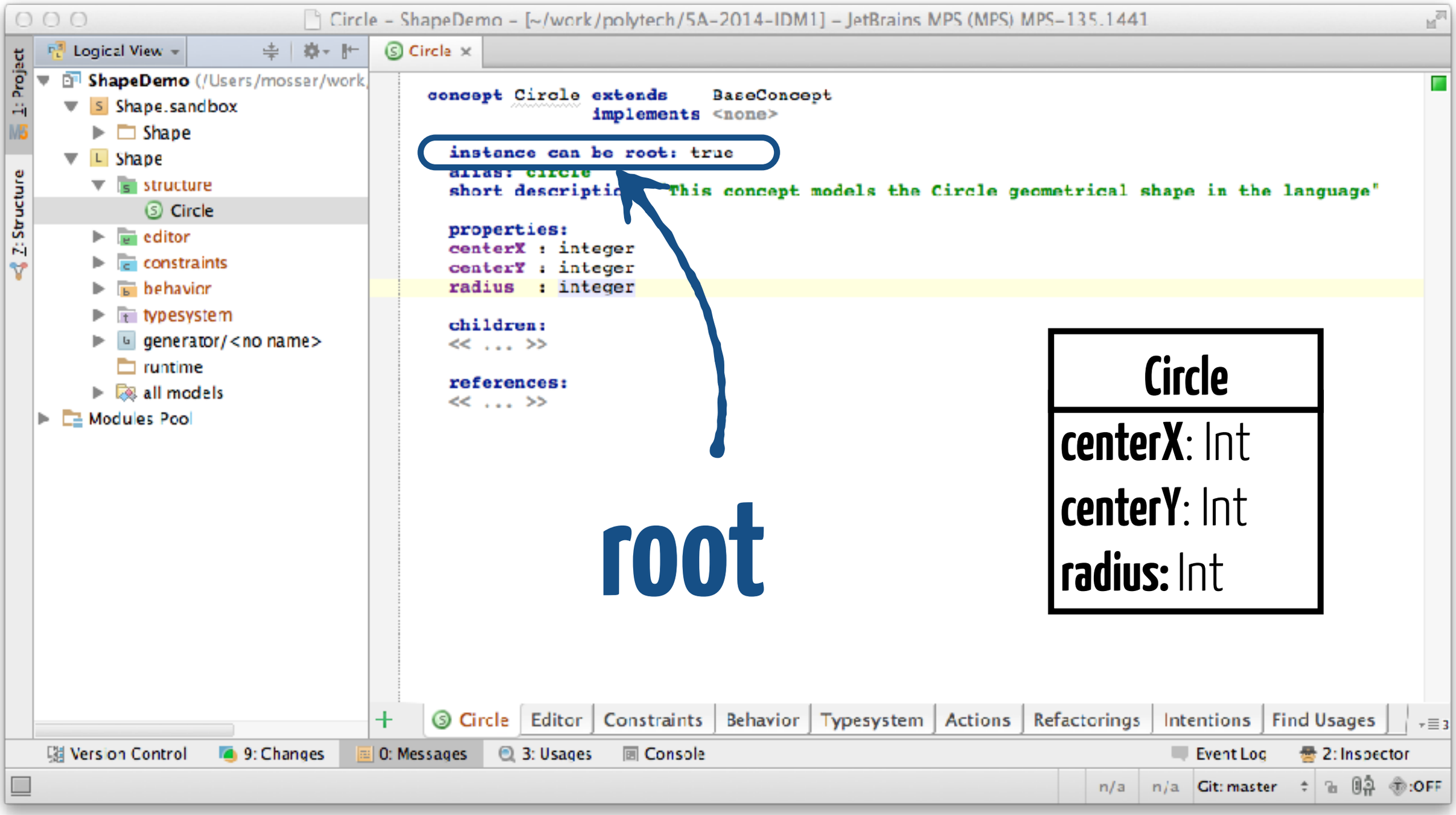
conforms to

Modeling

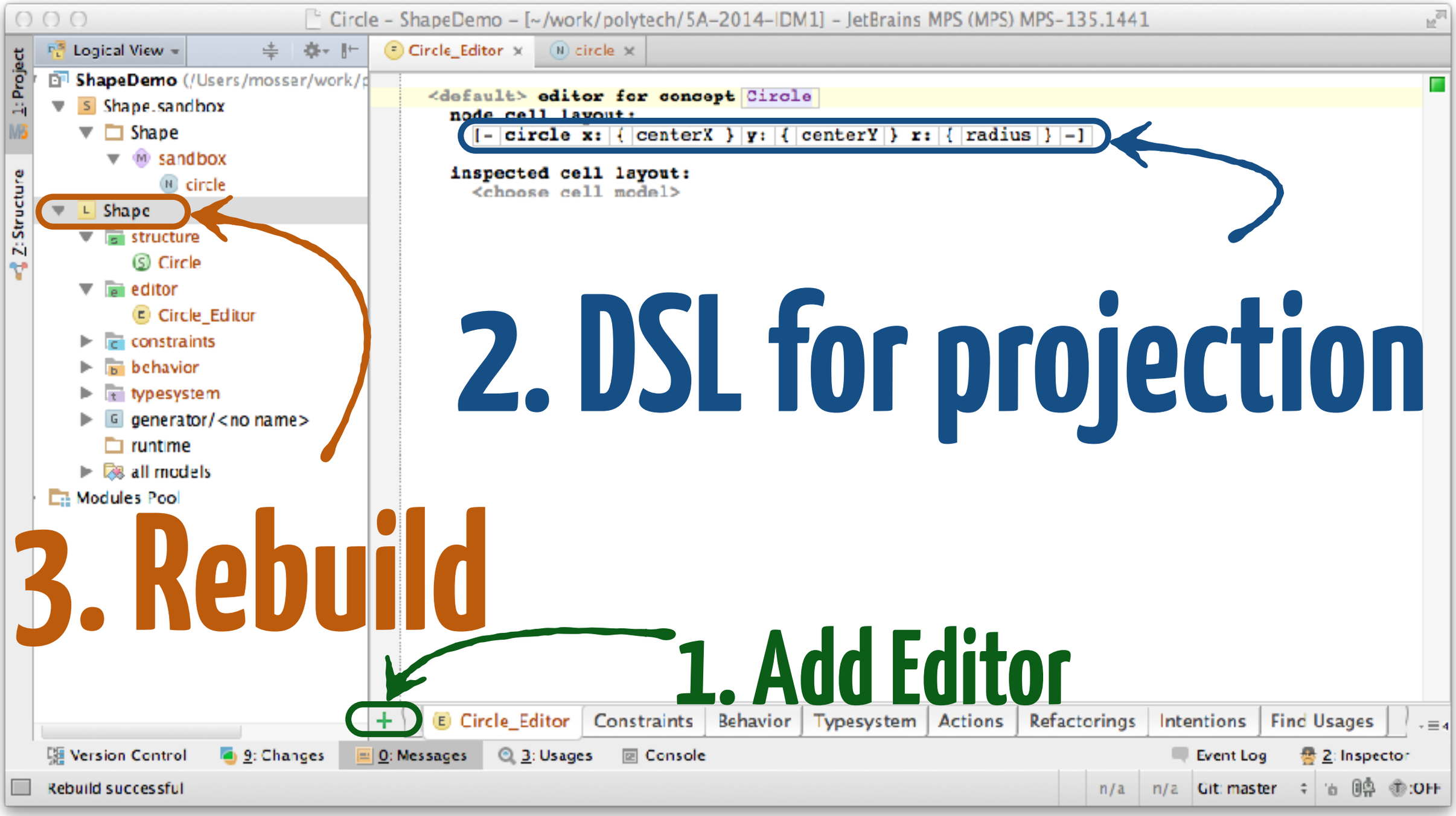
Environment



Meta-modeling a "Circle" (concept)



Defining a **projection** for Circle



Project(**AST**) ↦ Syntax

: Circle
centerX = 0
centerY = 0
radius = 200

```
[- circle x: { centerX } y: { centerY } r: { radius } -]
```

↦

```
circle x: 0 y: 0 r: 200
```

```
[/
    {
    [- ---> "kind" : "circle" -]
    [- ---> "x" : { centerX } -]
    [- ---> "y" : { centerY } -]
    [- ---> "r" : { radius } -]
    }
/]
```

↦

```
{
   "kind" : "circle"
   "x" : 0
   "y" : 0
   "r" : 200
}
```

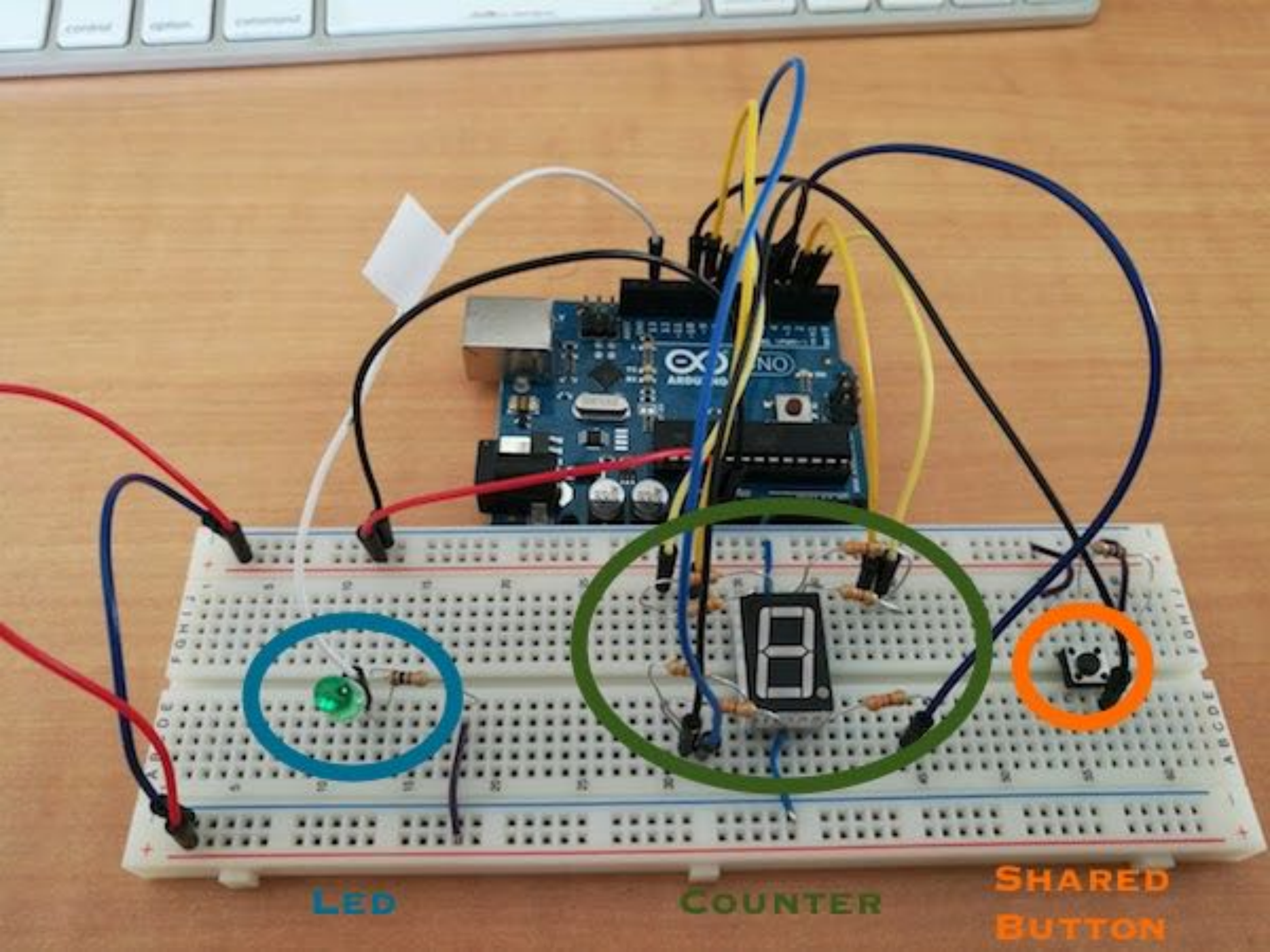
```
[- draw circle located at ( { centerX } , { centerY } ) with radius { radius } -]
```

↦

```
draw circle located at ( 0 , 0 ) with radius 200
```

Lab

Overview


Led
Counter
Shared
Button

1. Plain old C code
2. Using the ArduinoLib
3. Implementing a Finite State Machine
4. Modelling a Finite state machine
5. Using an existing language: Lustre
6. Implementing an external compiler
7. Embedding inside an host language
8. Using a DSL workbench

Step #1: Plain old C code

```
void change_led_state(){
  PORTB ^= 0b00100000; // Change digital 13 on->off->on (xor is life!)
}
```

```
void init(void)
{
  // DDRB is the configuration register for digital 7 to 18
  DDRB |= 0b00100000;// Digital 13 "outputmode"
  // TODO : enable write for digital 1 to 7 (7seg)
  // TODO : initialize global state values

}
```

```
int main(void)
{
  init();
  while(1) //infinite loop
    {
      // display_7seg(0); // uncomment to test the 7-seg when DDRD is configured
      change_led_state();
      _delay_ms(1000);     // 1Hz period
    }

  return 0;
}
```

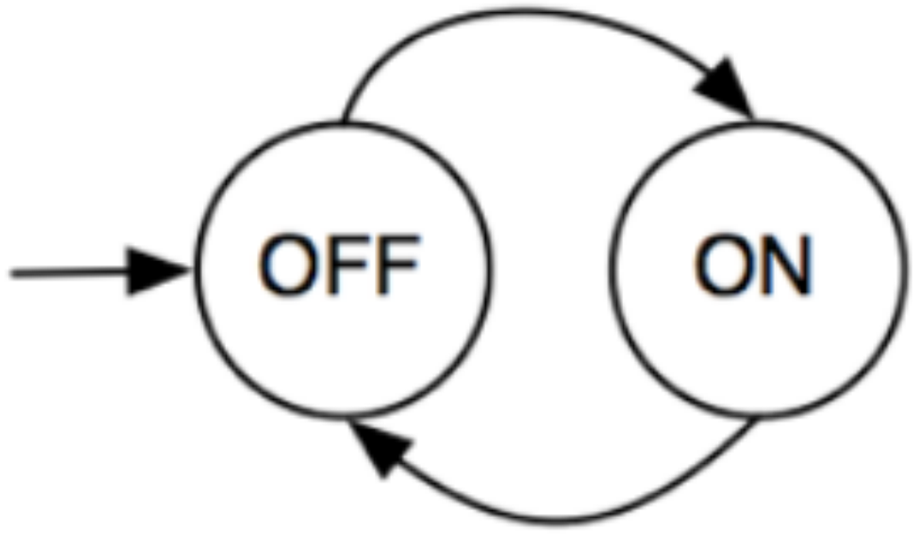
Step #2: Using the Arduino Library

```
int main(void)
{
  setup();
  while(1)
    {
      // displayDigit(0);
      change_state_led();
      _delay_ms(1000);

    }
  return 0;
}
```

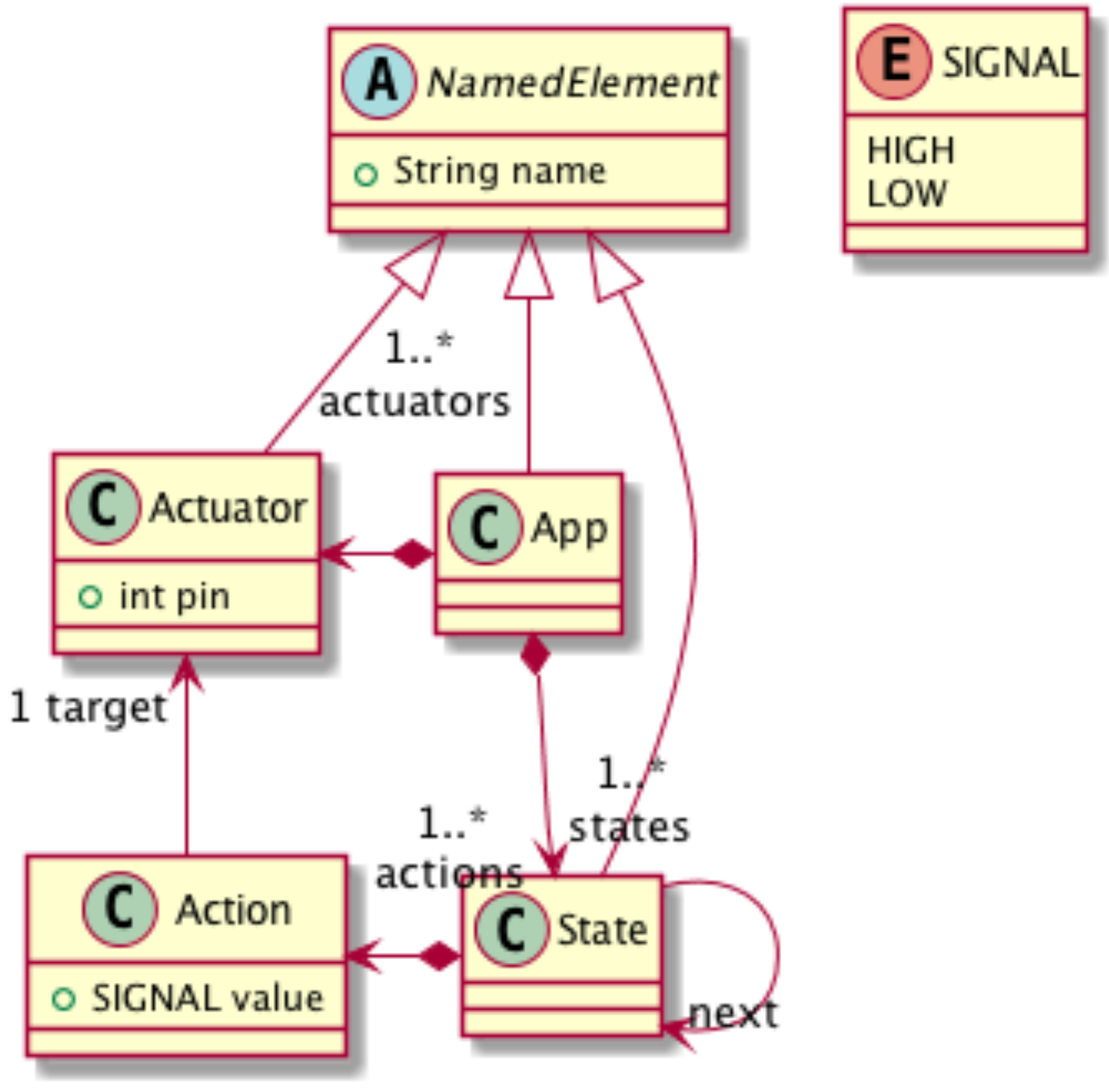
```
void change_state_led(){
  if (led_on){
    digitalWrite(led, LOW);
  } else
    digitalWrite(led, HIGH);
  led_on = !led_on;
}
```

Step #3: Implementing an FSM

```
int main(void)
{
  setup();
  state_on(); // initial state
  return 0;
}


void state_on() {
  digitalWrite(led, HIGH);
  _delay_ms(1000);
  state_off();
}

void state_off() {
  digitalWrite(led, LOW);
  _delay_ms(1000);
  state_on();
}
```

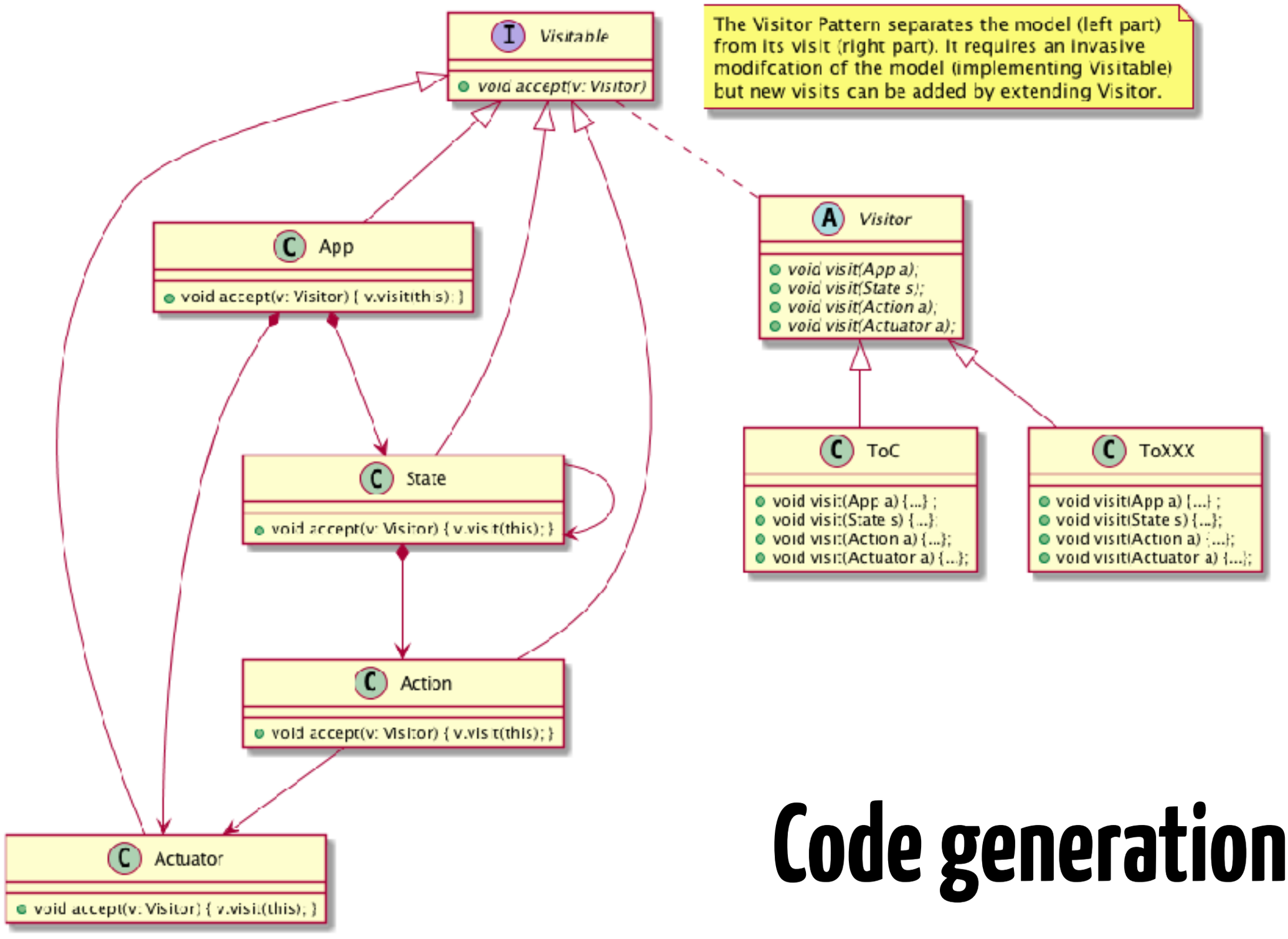


Step #4: Modelling an FSM



Meta-model

Step #4: Modelling an FSM

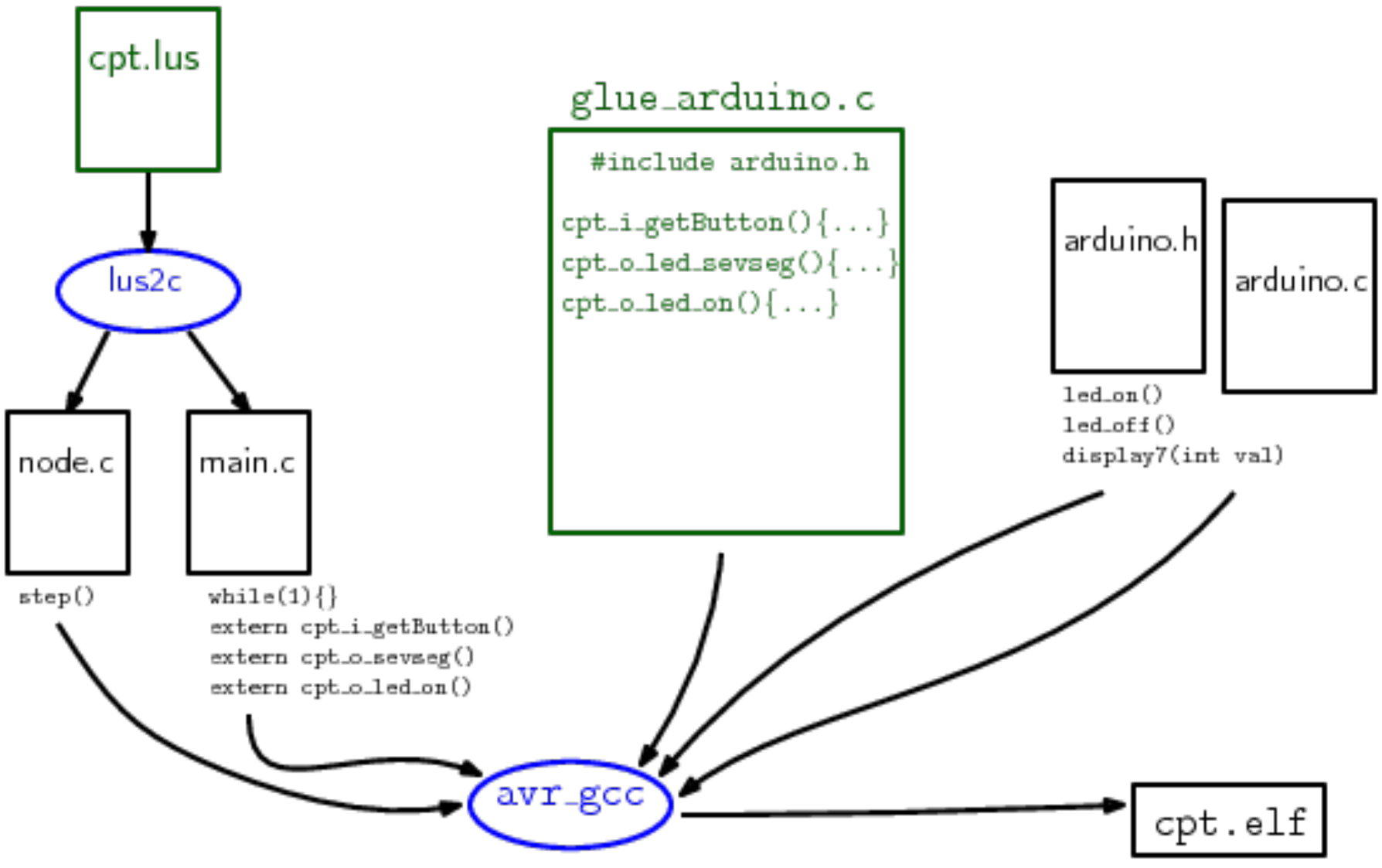


Code generation

Step #5: Using a dedicated language

```
node cpt(reset: bool)  returns (led_on: bool) ;
let
    led_on = false -> not(pre(led_on));
tel
```

Step #5: Using a dedicated language



Step #6: Using an external DSL

```
app         :   'application' name=IDENTIFIER '{' actuator+ state+ '}';

actuator    :   'actuator' location ;
location    :   id=IDENTIFIER ':' port=PORT_NUMBER;

state       :   initial? name=IDENTIFIER '{'  action+ next '}';
action      :   receiver=IDENTIFIER 'is' value=SIGNAL;
next        :   'goto' target=IDENTIFIER ;
initial     :   '->';
```

Step #6: Using an external DSL

```
@Override public void enterState(ArduinoMLParser.StateContext ctx) {
        State local = new State();
        local.setName(ctx.name.getText());
        this.currentState = local;
        this.states.put(local.getName(), local); // Symbol table for states
}

@Override public void exitState(ArduinoMLParser.StateContext ctx) {
        this.theApp.getStates().add(this.currentState);
        this.currentState = null;
}
```

Step #7: Using an embedded DSL

```
application("theLed")
        .uses(actuator("led", 13))

        .hasForState("on")
                .setting("led").toHigh()
                .goingTo("off")

        .hasForState("off")
                .initial()
                .setting("led").toLow()
                .goingTo("on")

        .export("./output/fsm.h", "./output/main.c");
```

Step #8: Using a Language Workbench

App ×

```
concept App extends     BaseConcept
            implements INamedConcept

  instance can be root: true
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  states     : State[1..n]
  actuators  : Actuator[1..n]

  references:
  << ... >>
```

Create Editor | What's this?

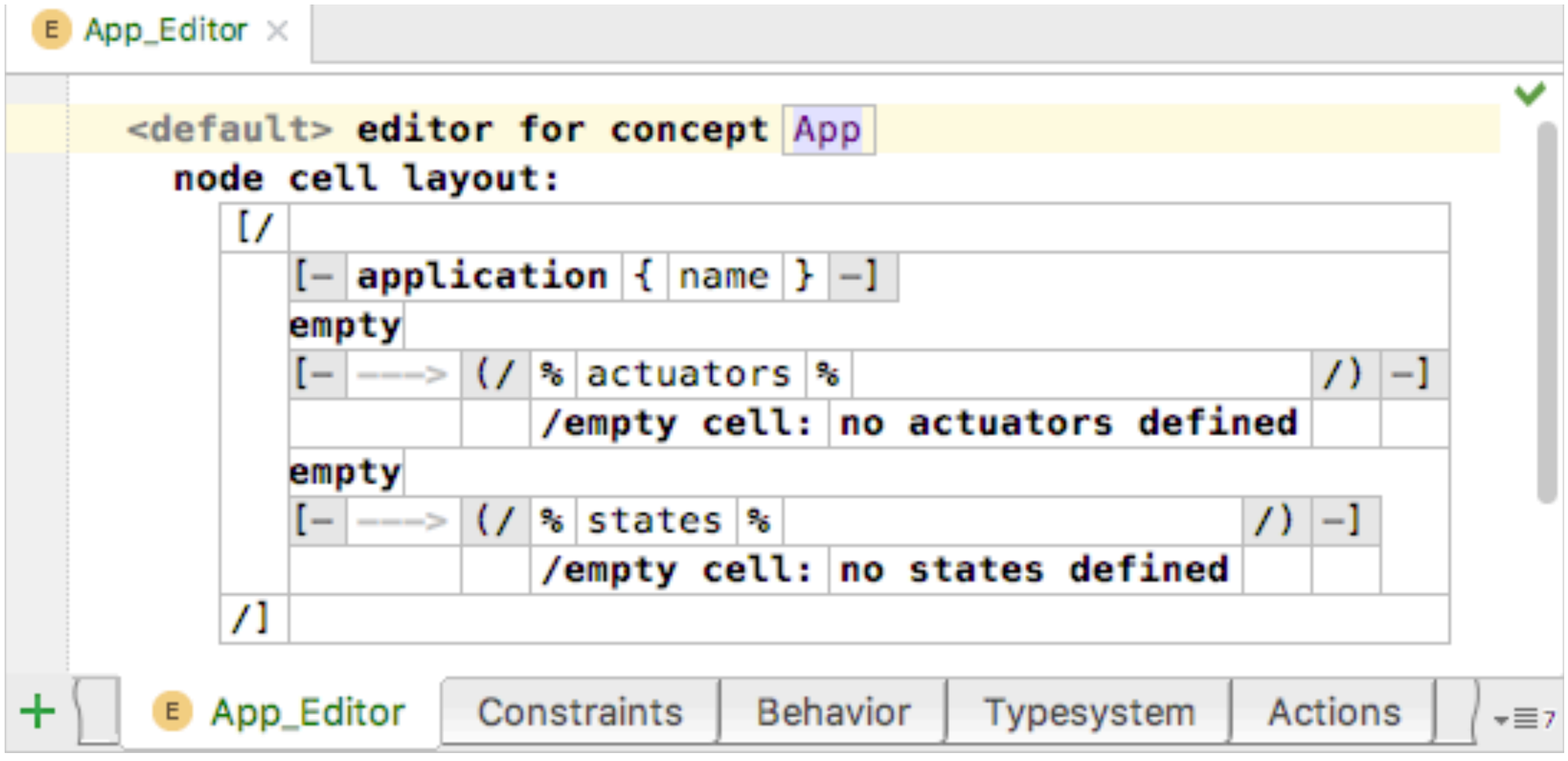
App | Editor | Constraints | Behavior

Step #8: Using a Language Workbench

```
app LED {

  states :
    state on next : off {
      is initial : false

      actions :
        action target : theLed {
          signal : high

        }
    }
    state off next : on {
      is initial : true

      actions :
        action target : theLed {
          signal : low

        }
    }
  actuators :
    actuator theLed {
      pin : 13

    }
}
```

```
application led

  actuator: theLed on pin 13

  on :       initial: true
    theLed <= high
    going to off

  off :      initial: false
    theLed <= low
    going to on
```

Step #8: Using a Language Workbench



Step #8: Using a Language Workbench

```
checking rule unique_initial_state {
  applicable for concept = App as app
  overrides false

  do {
    if (app.states.where({~s => s.isInitial :eq: true; }).size > 1) {
      error "Duplicated initial state detected in" + app.name + "!" -> app;
    }
  }
}
```

Step #8: Using a Language Workbench

```
concept behavior App {

  constructor {
    <no statements>
  }

  public Scope getScope(concept<> kind, node<> child)
  overrides ScopeProvider.getScope {
  if (kind.isSubConceptOf(State)) {

    return SimpleRoleScope.forNamedElements(this, link/App : states/);
  }
  if (kind.isSubConceptOf(Actuator)) {
    return SimpleRoleScope.forNamedElements(this, link/App : actuators/);

  }
  return null;
}
```

Step #8: Using a Language Workbench

```
append {// Declaring states} \n;
append $list{node.states with \n};
append \n \n;
append {void setup()} \n {{} \n;
with indent {
  node.actuators.forEach({~it =>
    indent buffer;
    append {pinMode(} ${it.name} {, } {OUTPUT);} \n;
  });
}
```