Domain Specific Language

Intro

Julien Deantoni
Universite Cote d'Azur,
CNRS I3S, INRIA KAIROS
Julien.deantoni@univ-cotedazur.fr

Who never used a DSL ?

Who never used a DSL ?

```
 8 bool is_valid_char( char c, std::string::size_type pos )
 9 {
10     const char dash = '-' ;
11
12     if( pos == 3 || pos == 6 ) // positions where dash is expected
13         return c == dash ;
14
15     else // positions where a digit is expected
16         return std::isdigit(c) ;
17 }
18
19
20 bool is_phone_number( const std::string& candidate )
21 {
22     const std::string::size_type EXPECTED_SIZE = 3+1+3+1+4 ;
23
24     if( candidate.size() != EXPECTED_SIZE ) return false ;
25
26     for( std::size_t i = 0 ; i < EXPECTED_SIZE ; ++i ) // for each position in the string
27         if( !is_valid_char( candidate[i], i ) ) return false ;
28
29     return true ;
30 }
```

/^\d{3}-\d{3}-\d{4}$/

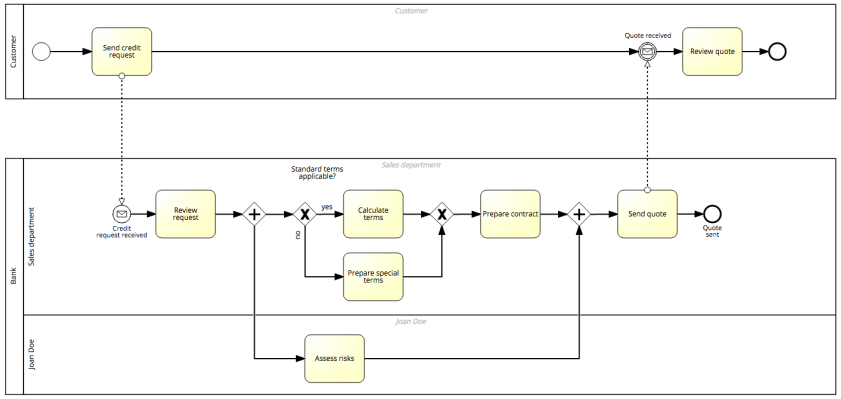
```
8  bool is_valid_char( char c, std::string::size_type pos )
9  {
10     const char dash = '-' ;
11
12     if( pos == 3 || pos == 6 ) // positions where dash is expected
13         return c == dash ;
14
15     else // positions where a digit is expected
16         return std::isdigit(c) ;
17 }
18
19
20 bool is_phone_number( const std::string& candidate )
21 {
22     const std::string::size_type EXPECTED_SIZE = 3+1+3+1+4 ;
23
24     if( candidate.size() != EXPECTED_SIZE ) return false ;
25
26     for( std::size_t i = 0 ; i < EXPECTED_SIZE ; ++i ) // for each position in the string
27         if( !is_valid_char( candidate[i], i ) ) return false ;
28
29     return true ;
30 }
```

DSL examples

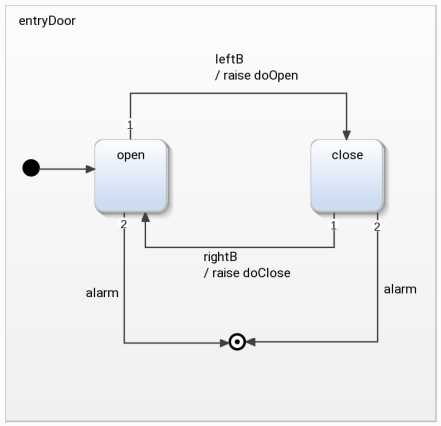
```
- name: "Demonstrate connecting to switches"
  hosts: switches
  gather_facts: no

  tasks:
    ###
    # Collect data
    #
    - name: Gather facts (eos)
      arista.eos.eos_facts:
      when: ansible_network_os == 'arista.eos.eos'

    - name: Gather facts (ios)
      cisco.ios.ios_facts:
      when: ansible_network_os == 'cisco.ios.ios'

    - name: Gather facts (vyos)
      vyos.vyos.vyos_facts:
      when: ansible_network_os == 'vyos.vyos.vyos'
```

```
h1 { color: white;
background: orange;
border: 1px solid black;
padding: 0 0 0 0;
font-weight: bold;
}
/* begin: seaside-theme */

body {
background-color:white;
color:black;
font-family:Arial,sans-serif;
margin: 0 4px 0 0;
border: 12px solid;
}
```

CSS




```
SELECT CustomerName, City FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```




```
CC = gcc
EXEC = prog
LIBFLAG = -L. -lmtm2
CFLAGS = -std=c99 -Wall -pedantic-errors -Werror

album.o: album.c records.h mtm_ex2.h set.h list.h album.h
list.o: list.c list.h
album_test.o: ./tests/album_test.c album.h
tests: album_test.o list.o album.o
    gcc $(CFLAGS) album_test.o list.o album.o -o $(EXEC) $(LIBFLAG)
```

```
Scenario Outline: Title of your scenario outline
  Given I want to write a step with <name>
  When I check for the <value> in step
  Then I verify the <status> in step

  Examples:
    | name  | value | status  |
    | name1 |     5 | success |
    | name2 |     7 | Fail    |
```

```
form->validateThat('password')->is()->not()->empty("You have not entered a password.")
 ->and()->longerThanOrEqualTo(8, "Your password must be at least 8 characters long.")
 ->and()->shorterThanOrEqualTo(50, "Your password is too long.")
 ->and()->identicalTo('passwordConfirm', "The passwords you have entered don't match.");
```

```
[^ ]*?@[^ ]*?\.[^ ]*
```

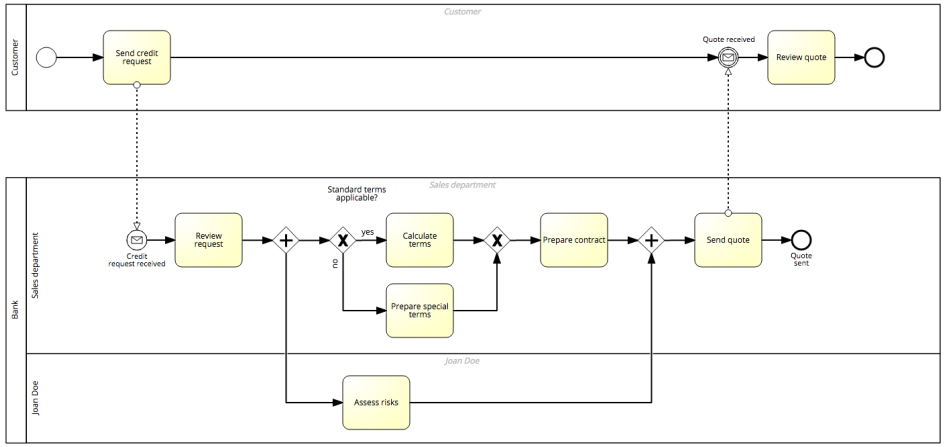
DSL examples

```
[^ ]*?@[^ ]*?\.[^ ]*
```

textual

```
h1 { color: white;
background: orange;
border: 1px solid black;
padding: 0 0 0 0;
font-weight: bold;
}
/* begin: seaside-theme */

body {
background-color:white;
color:black;
font-family:Arial,sans-serif;
margin: 0 4px 0 0;
border: 12px solid;
}
```

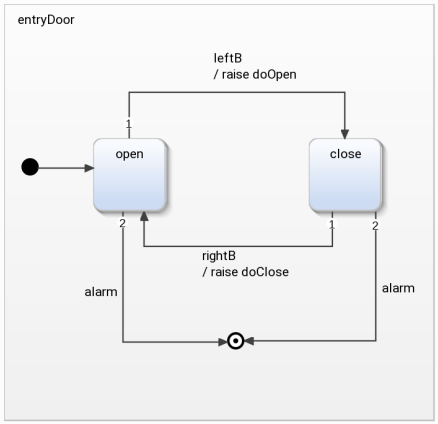
CSS



graphical

```
SELECT CustomerName, City FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

External DSL




```
CC = gcc
EXEC = prog
LIBFLAG = -L. -lmtm2
CFLAGS = -std=c99 -Wall -pedantic-errors -Werror

album.o: album.c records.h mtm_ex2.h set.h list.h album.h
list.o: list.c list.h
album_test.o: ./tests/album_test.c album.h
tests: album_test.o list.o album.o
    gcc $(CFLAGS) album_test.o list.o album.o -o $(EXEC) $(LIBFLAG)
```

```
form->validateThat('password')->is()->not()->empty("You have not entered a password.")
  ->and()->longerThanOrEqualTo(8, "Your password must be at least 8 characters long.")
  ->and()->shorterThanOrEqualTo(50, "Your password is too long.")
  ->and()->identicalTo('passwordConfirm', "The passwords you have entered don't match.");
```

Internal DSL

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; in a way that is clear, syntactically and semantically, to the reader being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; in a way that is clear, syntactically and semantically, to the reader being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, **usually of limited expressiveness**, used to specify (a part of) a system in a specific domain; in a way that is clear, syntactically and semantically, to the reader being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, **used to specify (a part of) a system in a specific domain**; in a way that is clear, syntactically and semantically, to the reader being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; **in a way that is clear**, syntactically and semantically, **to the reader** being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; **in a way that is clear**, **syntactically** and semantically, **to the reader** being either a programmer or a domain expert.

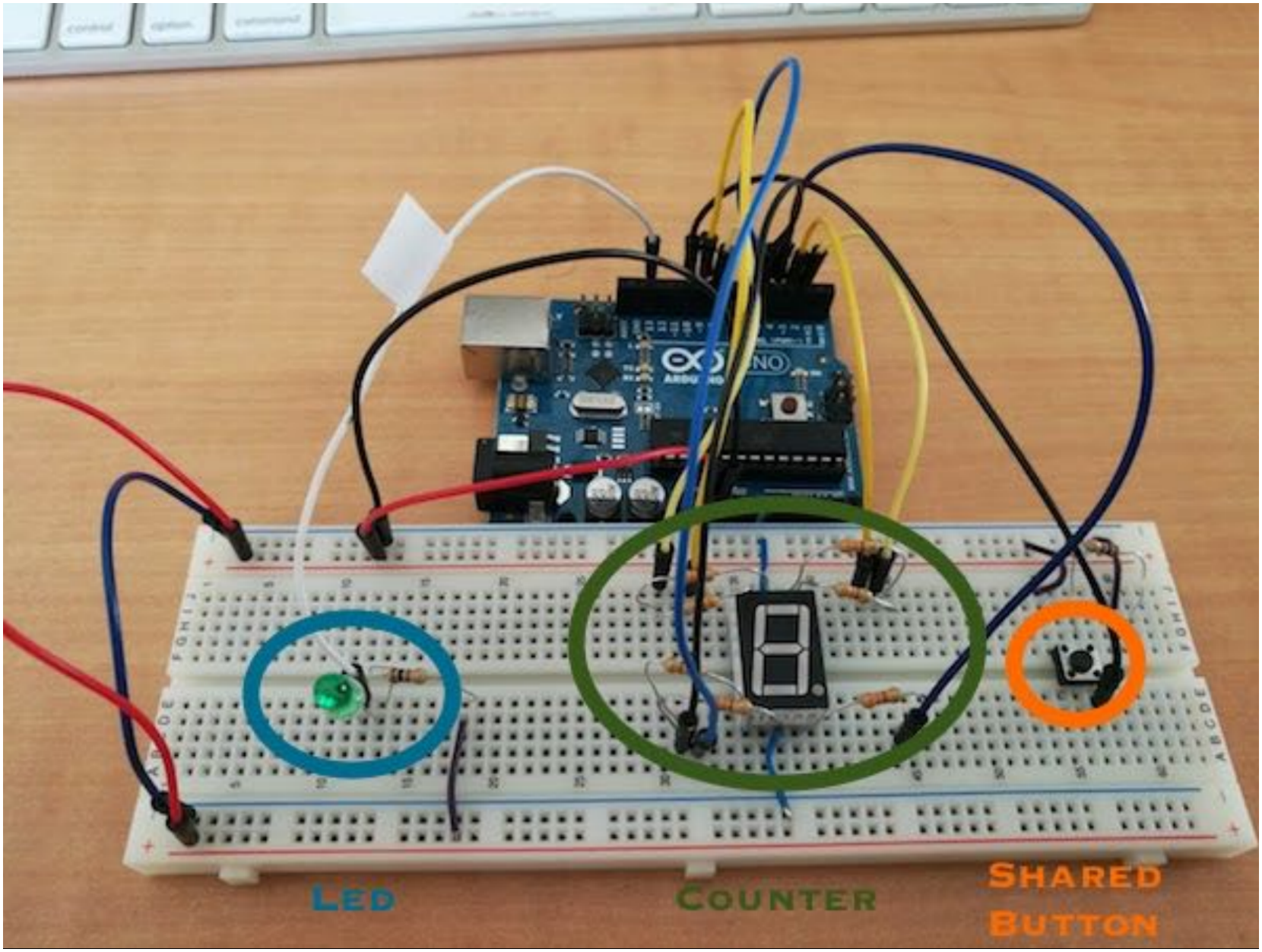
Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; **in a way that is clear**, syntactically and **semantically**, **to the reader** being either a programmer or a domain expert.

Definition (tentative)

A DSL is a computer language, usually of limited expressiveness, used to specify (a part of) a system in a specific domain; in a way that is clear, syntactically and semantically, to **the reader being either a programmer or a domain expert**.

Example from GPL to DSL



[Mosser]

Example from GPL to DSL

Step #1: Plain old C code

```
#include <avr/io.h>
#include <LiquidCrystal.h> // include a library headfile
#include <avr/interrupt.h>

unsigned char Delay(unsigned long a); // Active waiting Time delay function

unsigned short int run = 0;



int main(void)
{
cli(); »»   //disable global interrupts
DDRB = 0xFE;»   // Set PORTB as output but B0 as input -> 0b11111110
PCICR = 0x01;»  // choose PCINT[8..0]: here PCINT0 is enabled
PCMSK0 = 0x01;» // for now... PCINT0 is enabled (PCINT0 --> PB0 --> D8)

sei();» »   //enable interrupts

  while(1)
  {
»     if (run == 1){
»       PORTB = PORTB ^ 0x02;»  //switche PB1 --> D9
»     }
»     Delay(90000);»//wait a little
  }
}
```

```
unsigned char Delay (unsigned long a)
{
»   unsigned long b;
»   for(b=0;b<a;b++){
»   };
»   return b;
}

ISR ( PCINT0_vect )
{
  PCMSK0 = 0x00; »  »   //disable PCINT0
  if(run==0){
    run=1;
  }else{
    run=0;
  }
  Delay(100000);»   //stabilization delay
  PCMSK0 = 0x01; »  »   //enable PCINT0
}
```

11

[modified from Mosser]

Example from GPL to DSL

Step #2: Using the Arduino Library V1

```
const int LED_PIN = 13;
const int INTERRUPT_PIN = 2;
volatile bool ledState = LOW;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(INTERRUPT_PIN, INPUT_PULLUP);
    // trigger when button pressed, but not when released.
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), myISR, FALLING);
}

void loop() {
    digitalWrite(LED_PIN, ledState);
}

void myISR() {
    ledState = !ledState;
    // note: LOW == false == 0, HIGH == true == 1,
    //so inverting the boolean is the same as switching between LOW and HIGH.
}
```

[modified from Mosser]

Example from GPL to DSL

Step #2: Using the Arduino Library V2

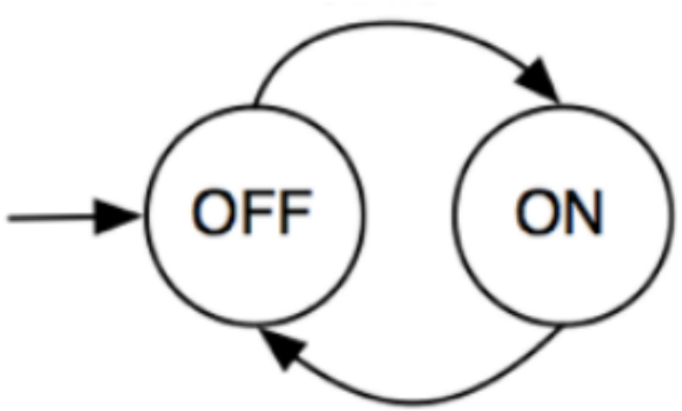
```
#include <stdio.h>
#include <stdlib.h>

const int button = 8;             // GPIO 8 for the button
const int led =7;                 // GPIO 7 for the LED
int ledflag=0;                    // LED status flag

void setup() {
  pinMode(button,INPUT);          // define button as an input
  pinMode(led,OUTPUT);            // define LED as an output
  digitalWrite(led,LOW);          // turn output off just in case
}

void loop() {
  if (digitalRead(button)==HIGH){ // if button is pressed
    if (ledflag==0) {              // and the status flag is LOW
      ledflag=1;                   // make status flag HIGH
      digitalWrite(led,HIGH);      // and turn on the LED
      }                            //
    else {                         // otherwise...
      ledflag=0;                   // make status flag LOW
      digitalWrite(led,LOW);       // and turn off the LED
    }
  delay(1000);                     // wait a sec for the
  }                                // hardware to stabilize
}                                  // begin again
```

[modified from Mosser]

Example from GPL to DSL

Step #3: Implementing an FSM

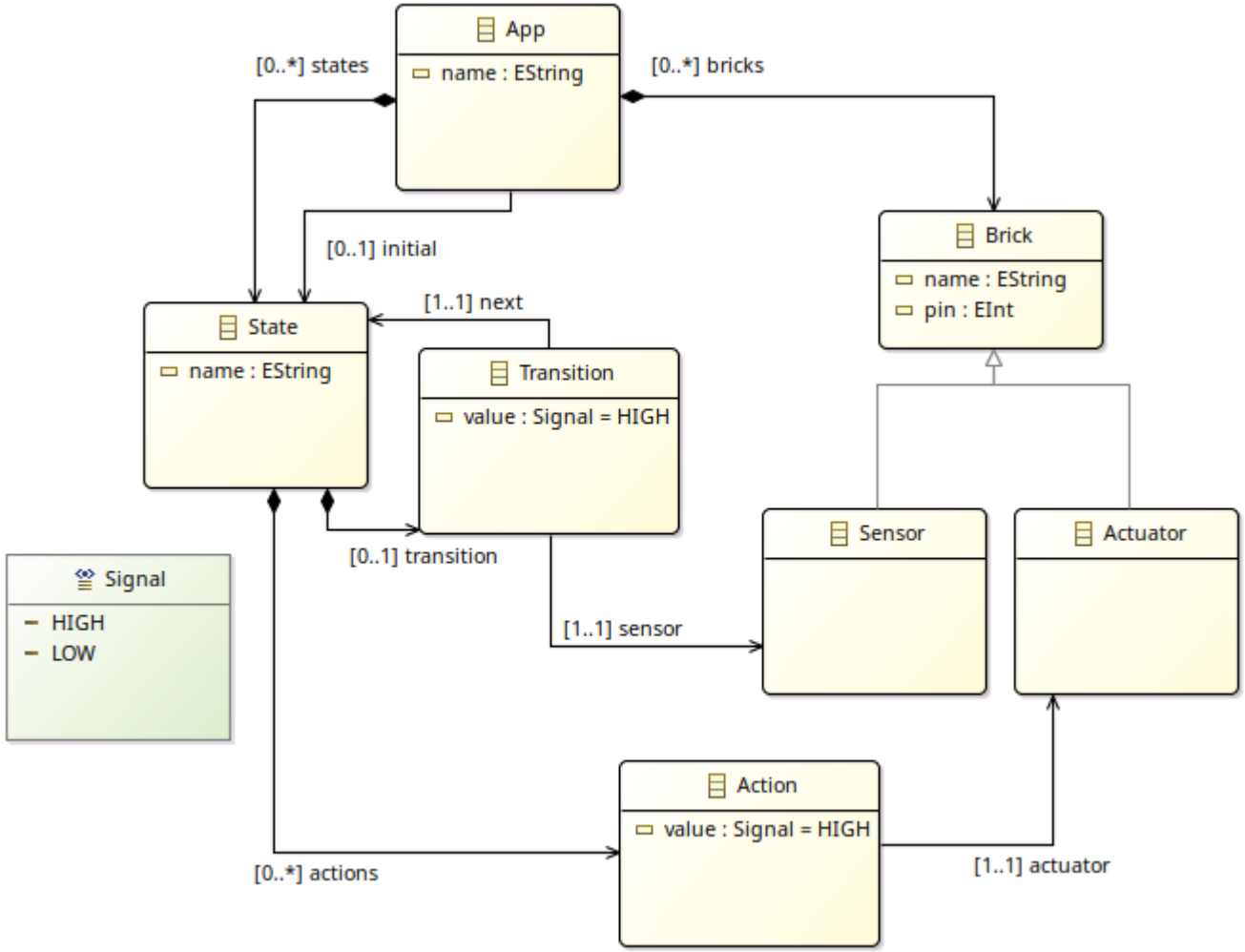
```
app RedButton initial state off {
    bricks
        Actuator red_led : 12
        Sensor button : 8

    states
        off {
            red_led <= LOW
            button is HIGH => on
        }
        on {
            red_led <= HIGH
            button is HIGH => off
        }
}
```



[modified from Mosser]

Example from GPL to DSL

Step #4: Modelling an FSM



Meta Model

[Mosser]

https://github.com/cucumber/gherkin

Example from GPL to DSL

External DSL

```
app RedButton initial state off {
    bricks
        Actuator red_led : 12
        Sensor button : 8

    states
        off {
            red_led <= LOW
            button is HIGH => on
        }
        on {
            red_led <= HIGH
            button is HIGH => off
        }
}
```

Internal DSL / Embedded DSL / Fluent API

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

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

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

Example from GPL to DSL

External DSL

```
app RedButton initial state off {
    bricks
        Actuator red_led : 12
        Sensor button : 8

    states
        off {
            red_led <= LOW
            button is HIGH => on
        }
        on {
            red_led <= HIGH
            button is HIGH => off
        }
}
```

Internal DSL / Embedded DSL / Fluent API

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

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

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

Pros:

- Harder to say wrong,
- easier to see when there is an error (even by business people)
- Awkward 3rd party library → DSL can make them very manageable
- Taking opportunity of declarative style

Cons:

- Language cacophony ?
- Ghetto language
- Wrong understanding of the DSL due to semantic variation points in stakeholders head

Example from GPL to DSL

External DSL

```
app RedButton initial state off {
    bricks
        Actuator red_led : 12
        Sensor button : 8

    states
        off {
            red_led <= LOW
            button is HIGH => on
        }
        on {
            red_led <= HIGH
            button is HIGH => off
        }
}
```

Internal DSL / Embedded DSL / Fluent API

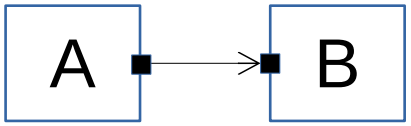
```
application("theLed")
        .uses(actuator("led", 13))

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

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

Pros:

- Harder to say wrong,
- easier to see when there is an error (even by business people)
- Awkward 3rd party library → DSL can make them very manageable
- Taking opportunity of declarative style

Cons:

- Language cacophony ?
- Ghetto language
- Wrong understanding of the DSL due to semantic variation points in stakeholders head



Example from GPL to DSL

External DSL

```
app RedButton initial state off {
    bricks
        Actuator red_led : 12
        Sensor button : 8

    states
        off {
            red_led <= LOW
            button is HIGH => on
        }
        on {
            red_led <= HIGH
            button is HIGH => off
        }
}
```

Internal DSL / Embedded DSL / Fluent API

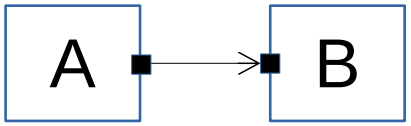
```
application("theLed")
        .uses(actuator("led", 13))

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

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

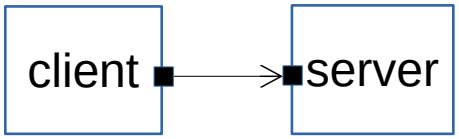
Pros:

- Harder to say wrong,
- easier to see when there is an error (even by business people)
- Awkward 3rd party library → DSL can make them very manageable
- Taking opportunity of declarative style

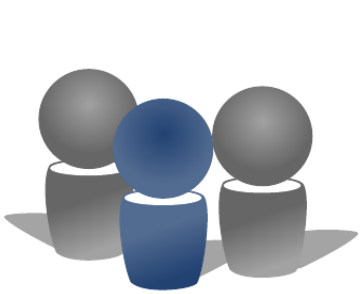
Cons:

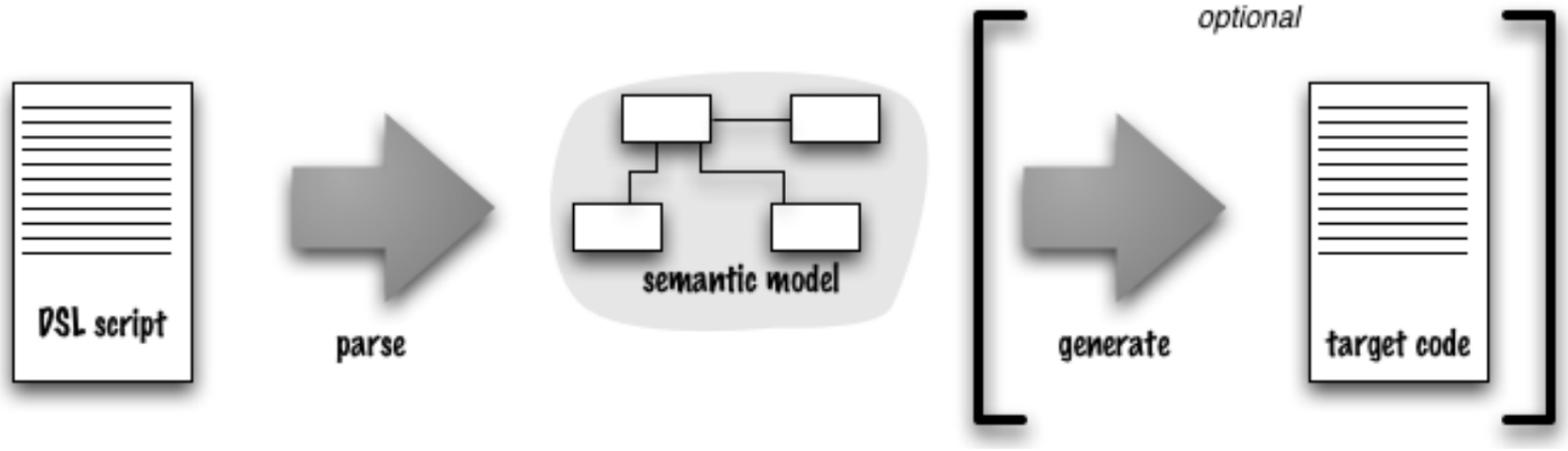
- Language cacophony ?
- Ghetto language
- Wrong understanding of the DSL due to semantic variation points in stakeholders head





DSL bird view and stakeholders

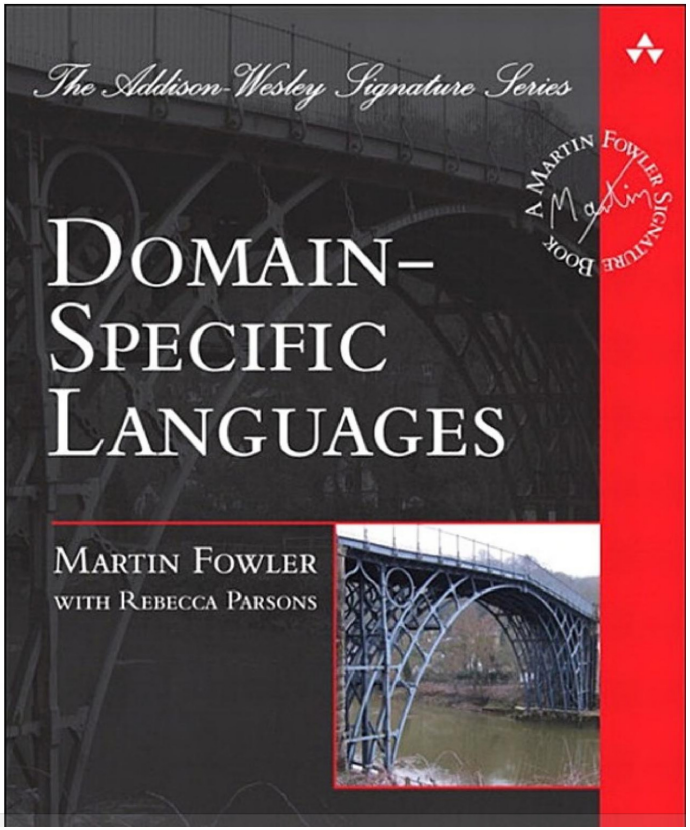


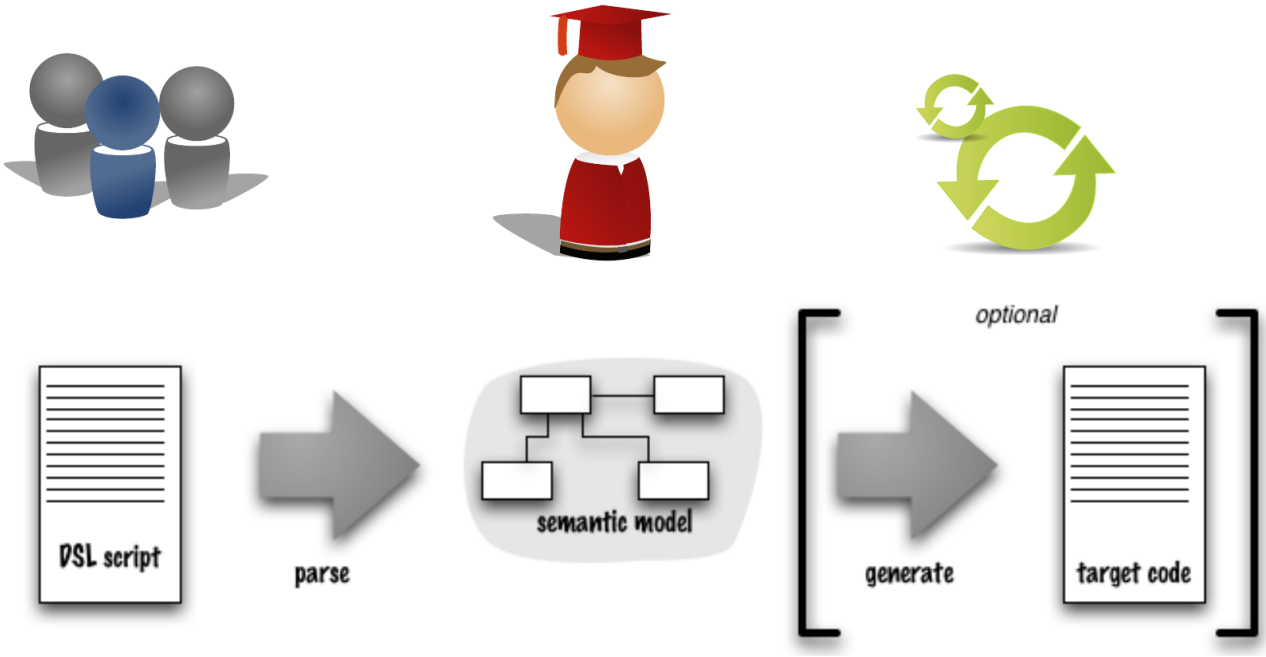


[Domain-Specific Languages]

[Mosser]

DSL bird view and stakeholders

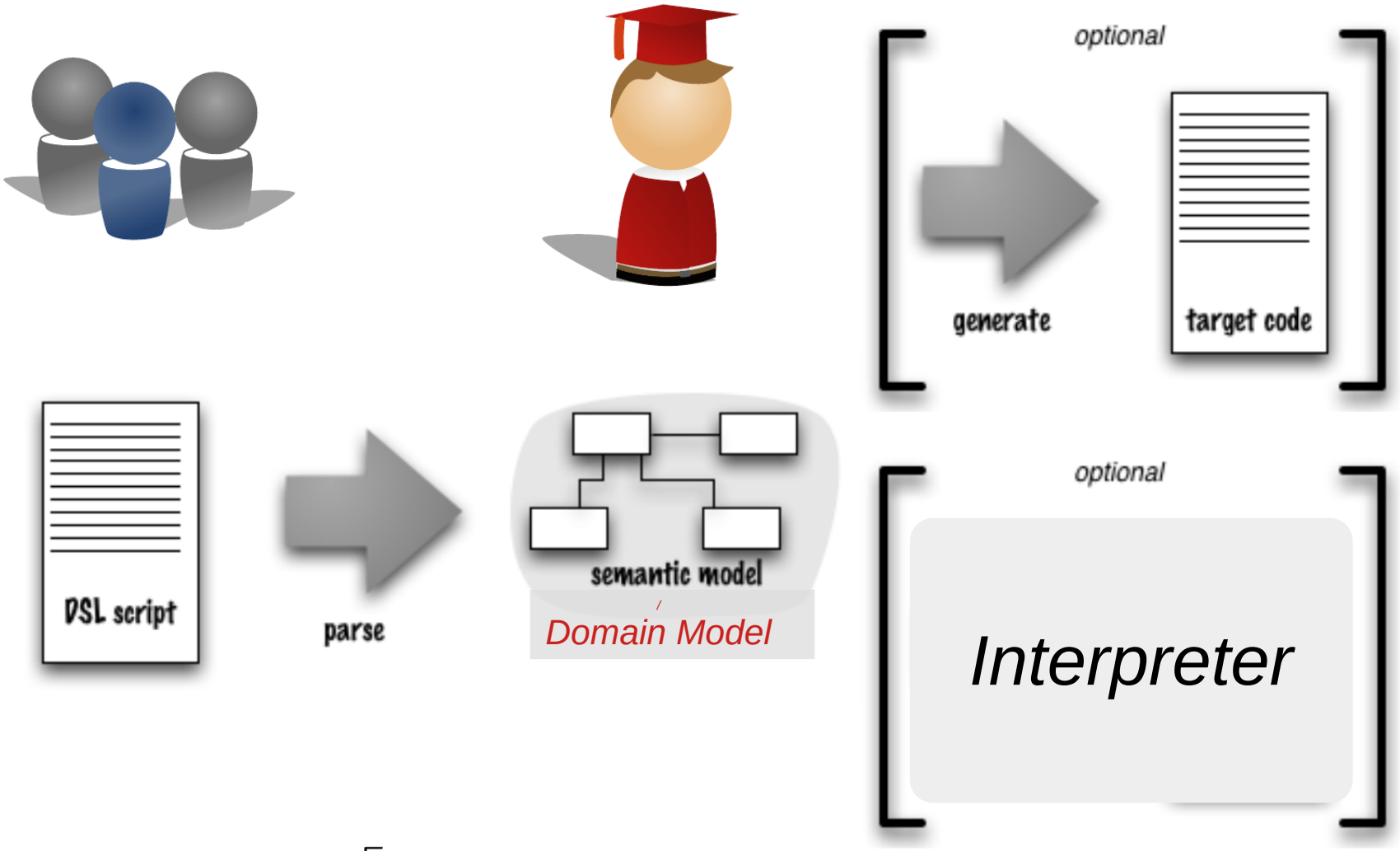




[Domain-Specific Languages]

[Mosser]

DSL bird view and stakeholders

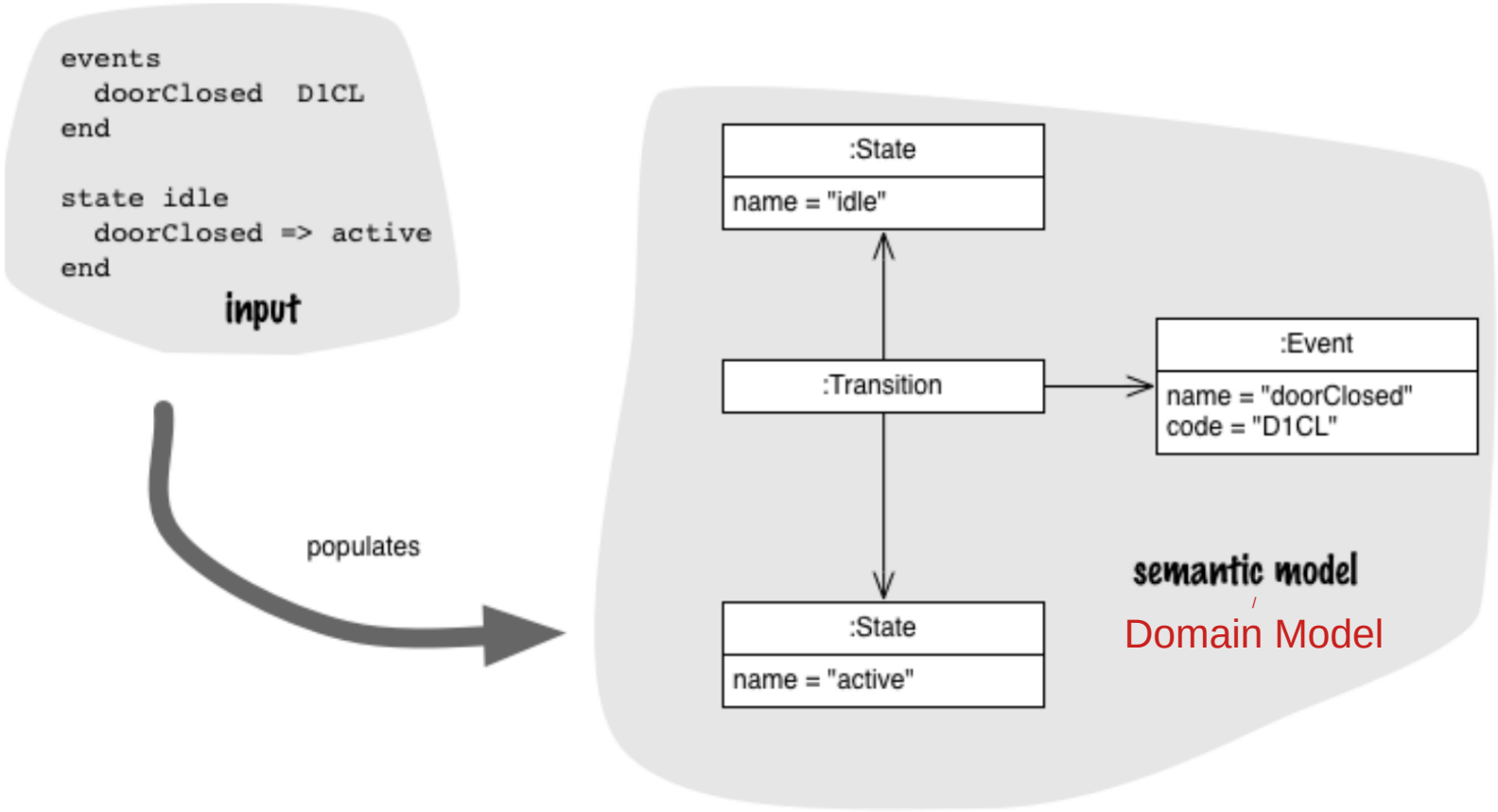


[Domain-Specific Languages]

[Mosser]

Modified

Semantic (or Domain) model

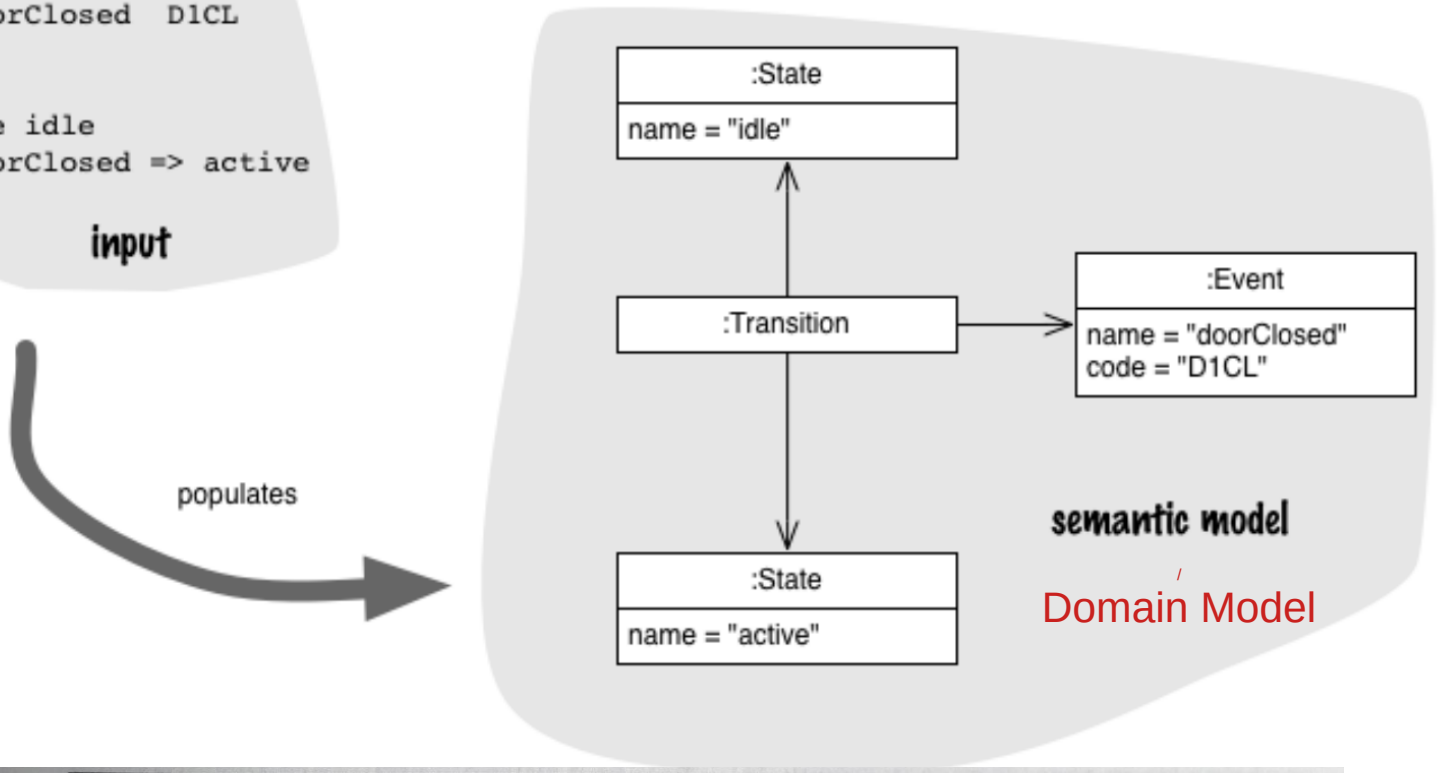


[Domain Specific Languages]

Semantic (or Domain) model

```
events
  doorClosed  D1CL
end

state idle
  doorClosed => active
end
```

input

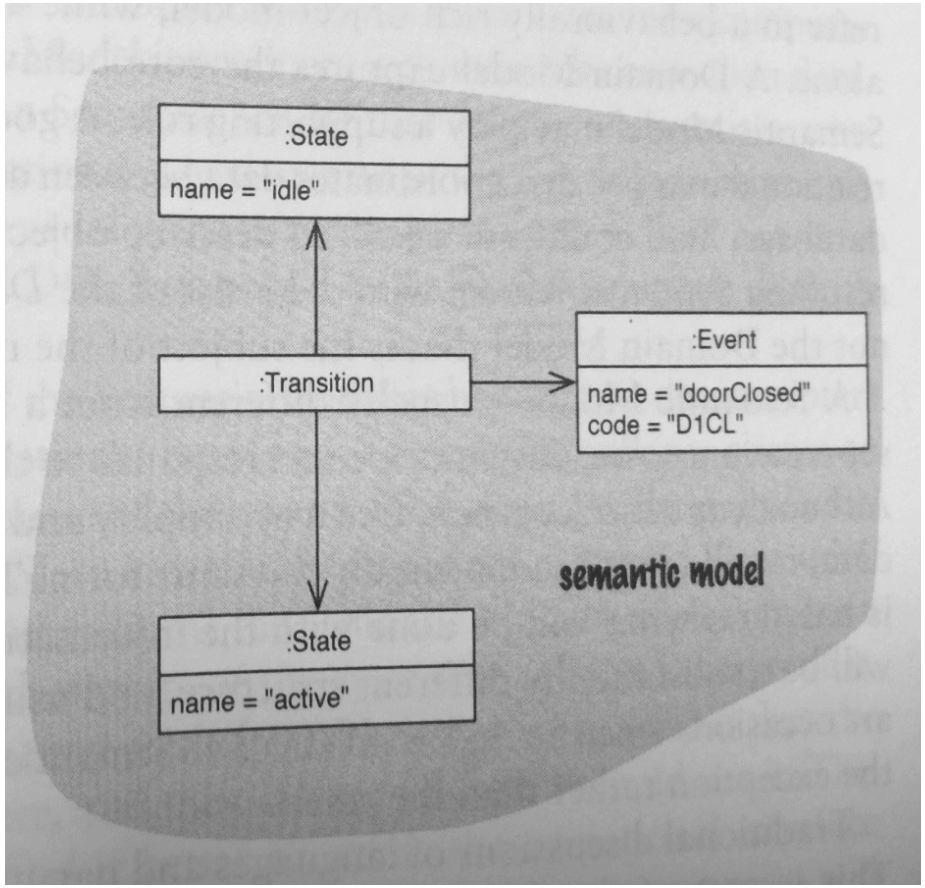


11.1 How It Works

[Domain Specific Languages]

In the context of a DSL, a semantic model is a representation, such as an in-memory object model, of the same subject that the DSL describes. If my DSL describes a state machine, then my Semantic Model might be an object model with classes for state, event, etc. A DSL script defining particular states and events would correspond to a particular population of that schema, with an event instance for each event declared in the DSL script. The Semantic Model is thus the library or framework that the DSL populates.

Semantic or Domain model ?

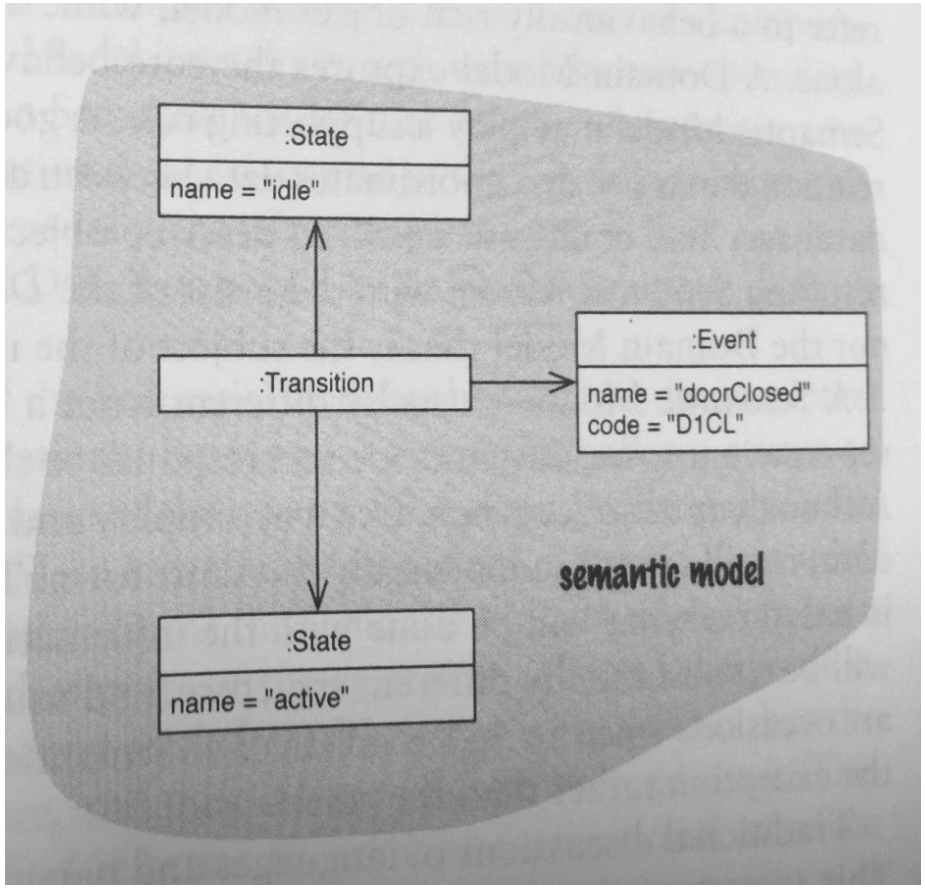


The term semantic model can be misleading

If the domain model is equipped with an interpreter or a way to be executed, then it acts as a semantic model since it defines the meaning of the DSL in terms of behavior.

If the domain model is a data structure (close to an AST), then it does not bring any information about the semantics in terms of behavior

Semantic or Domain model ?



The term semantic model can be misleading

If the domain model is equipped with an interpreter or a way to be executed, then it acts as a semantic model since it defines the meaning of the DSL in terms of behavior.

If the domain model is a data structure (close to an AST), then it does not bring any information about the semantics in terms of behavior

Warning: semantic is not understood here as in the ontology domain, but rather as a behavioral semantics (see trace semantics)

Alur, R., & Dill, D. (1990, July). Automata for modeling real-time systems. In International Colloquium on Automata, Languages, and Programming (pp. 322-335). Springer, Berlin, Heidelberg.

Semantic or Domain model ?