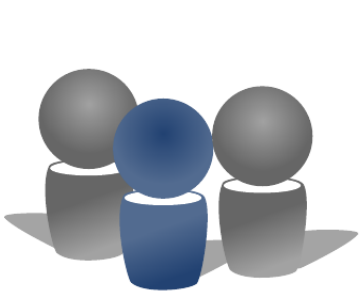
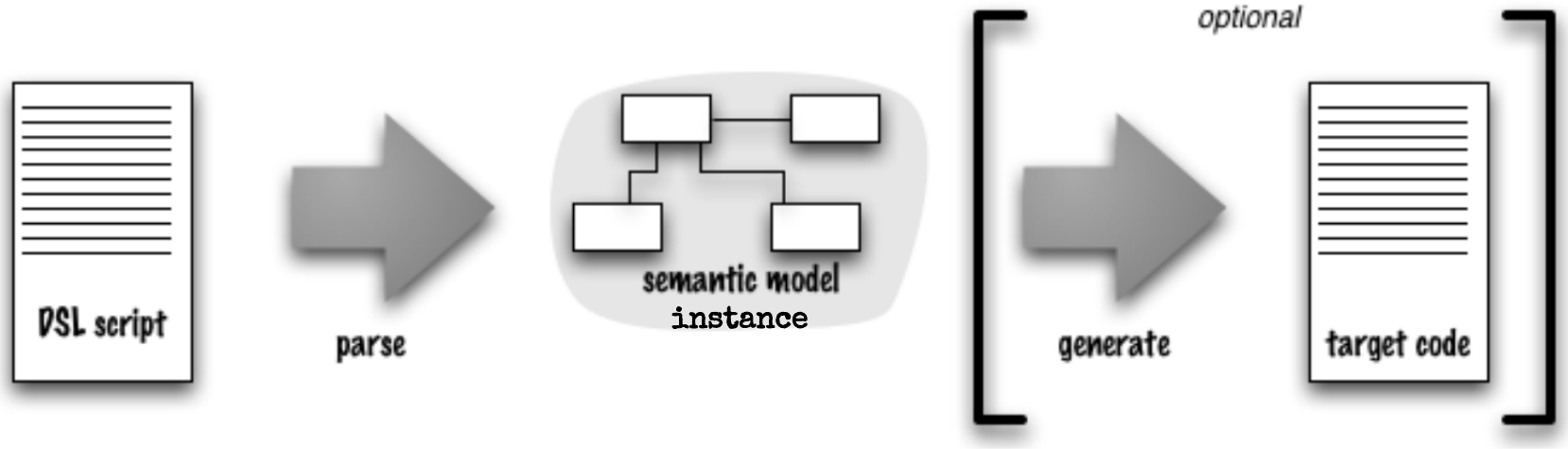
Notion of DSL and Behavioral Semantics of Languages

Julien Deantoni

DSL bird view and stakeholders

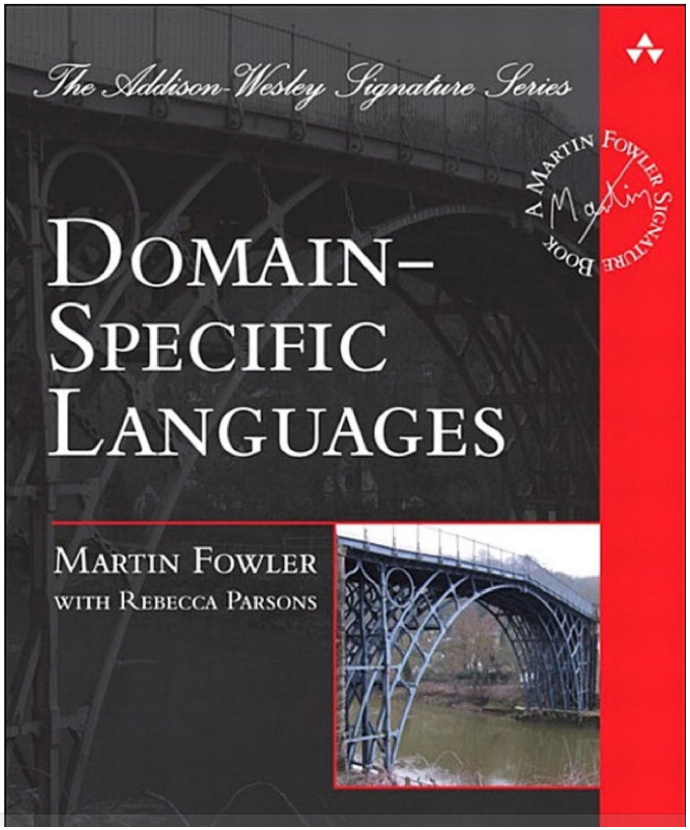


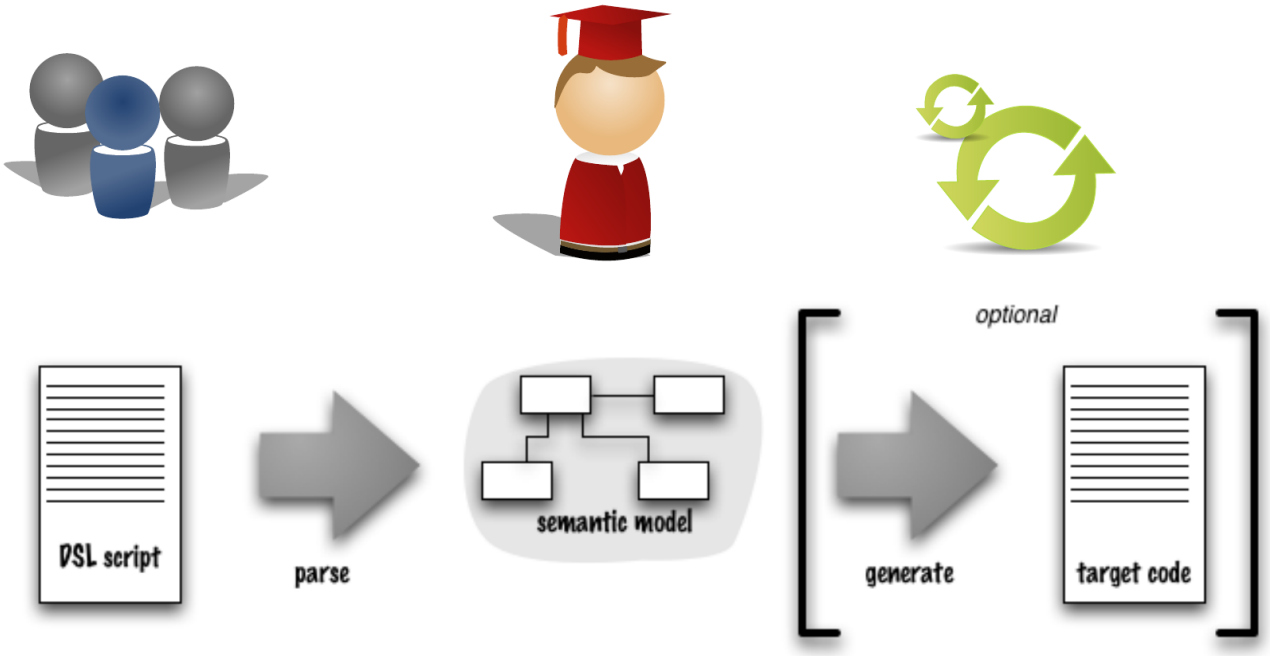


[Domain-Specific Languages]

Modified

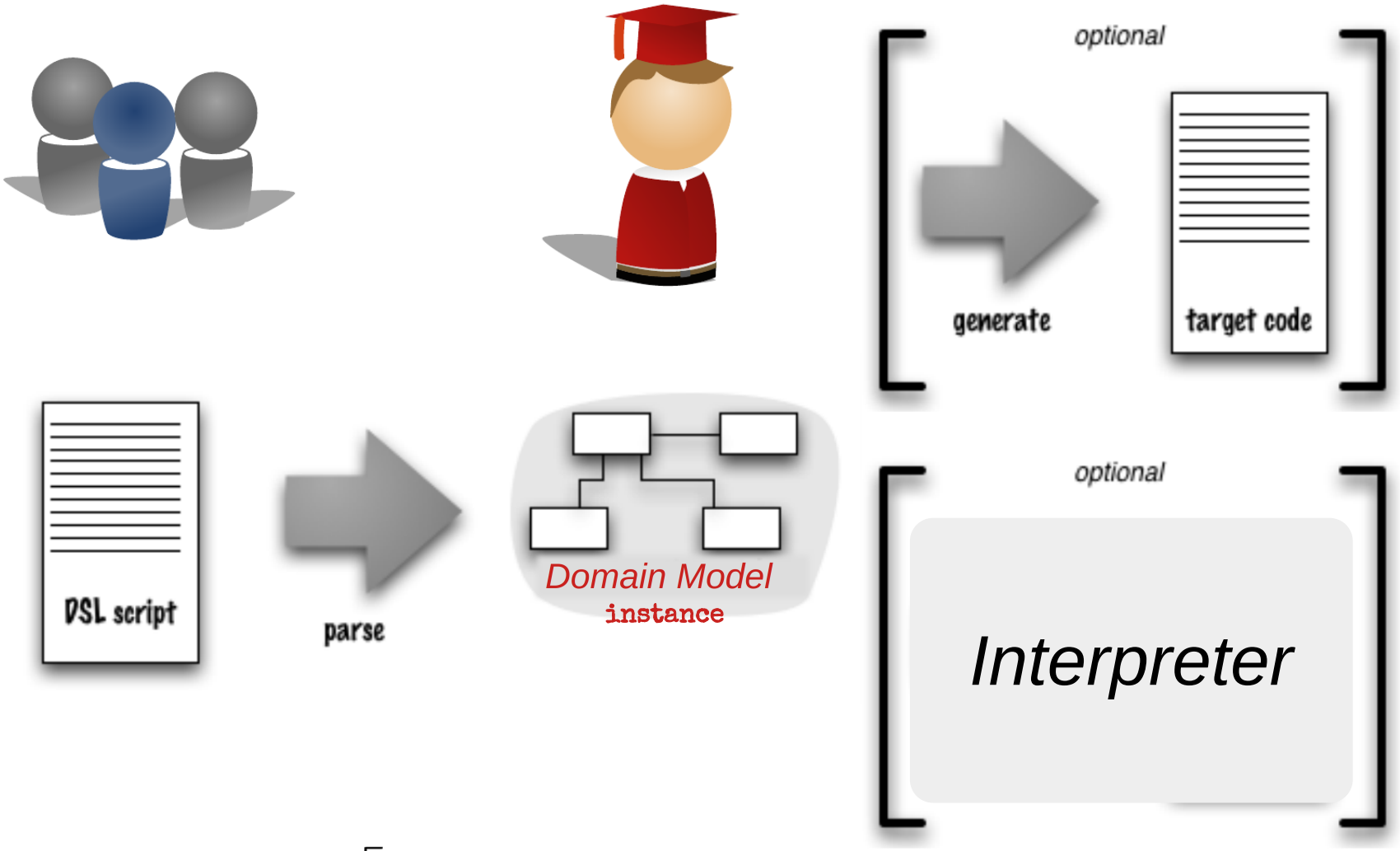
DSL bird view and stakeholders





[Domain-Specific Languages]

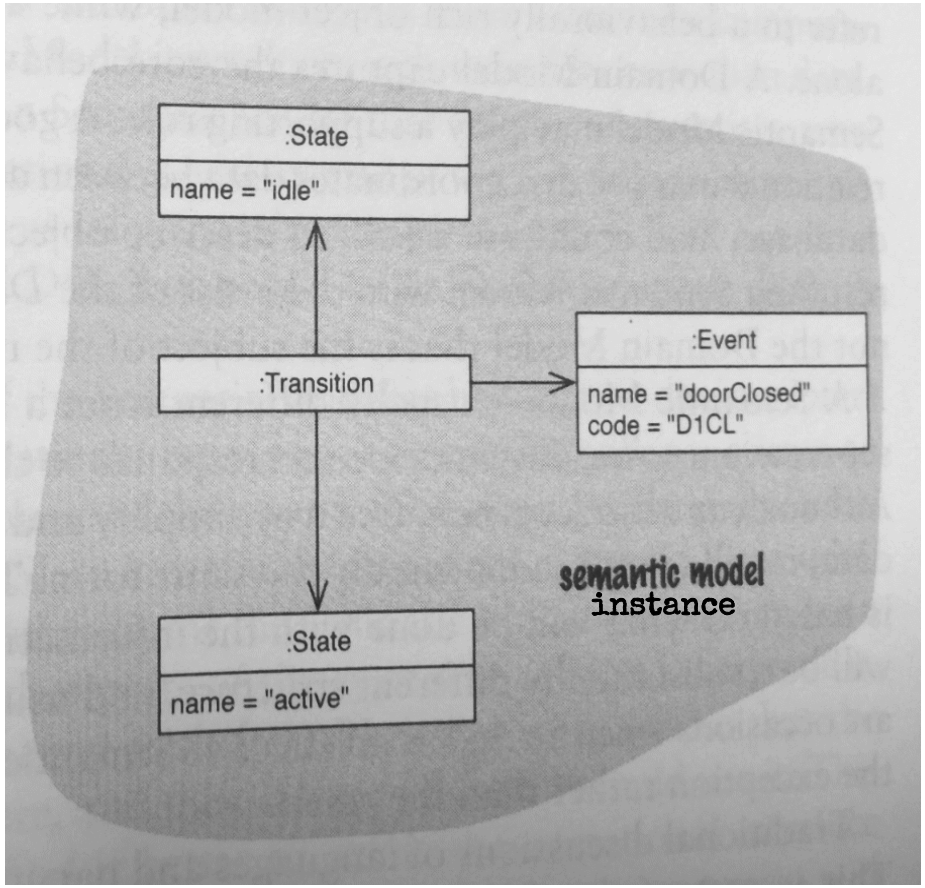
DSL bird view and stakeholders



[Domain-Specific Languages]

Modified

Semantic or Domain model ?

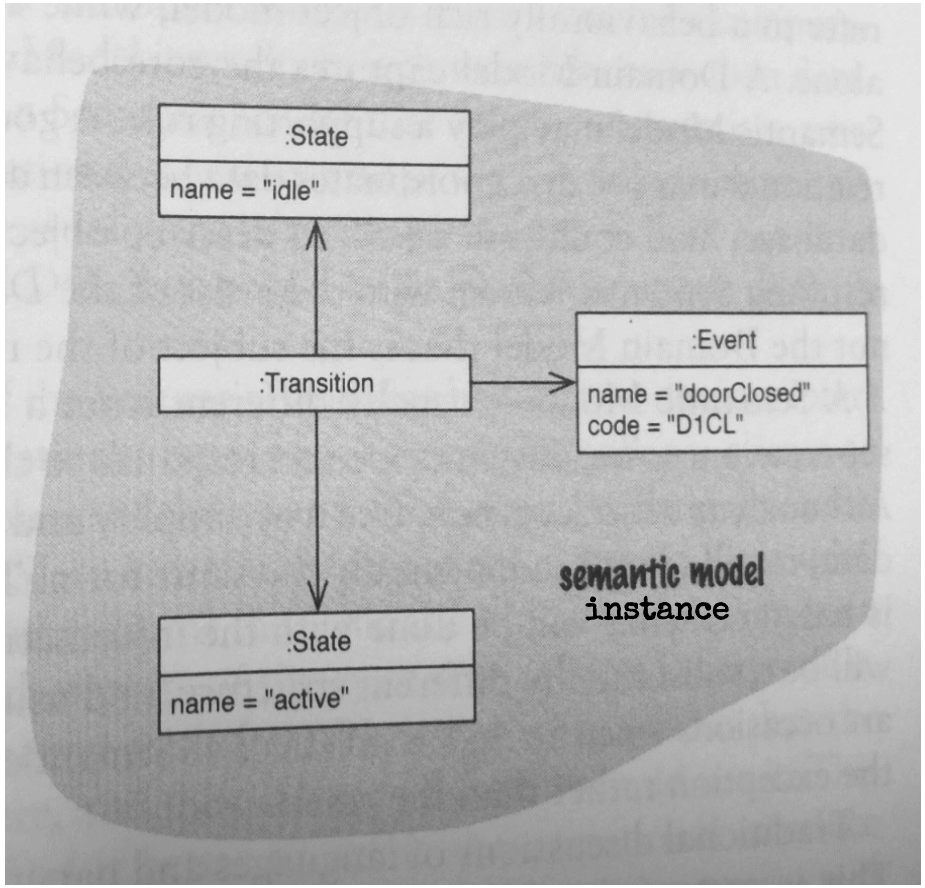


The term semantic model can be misleading

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

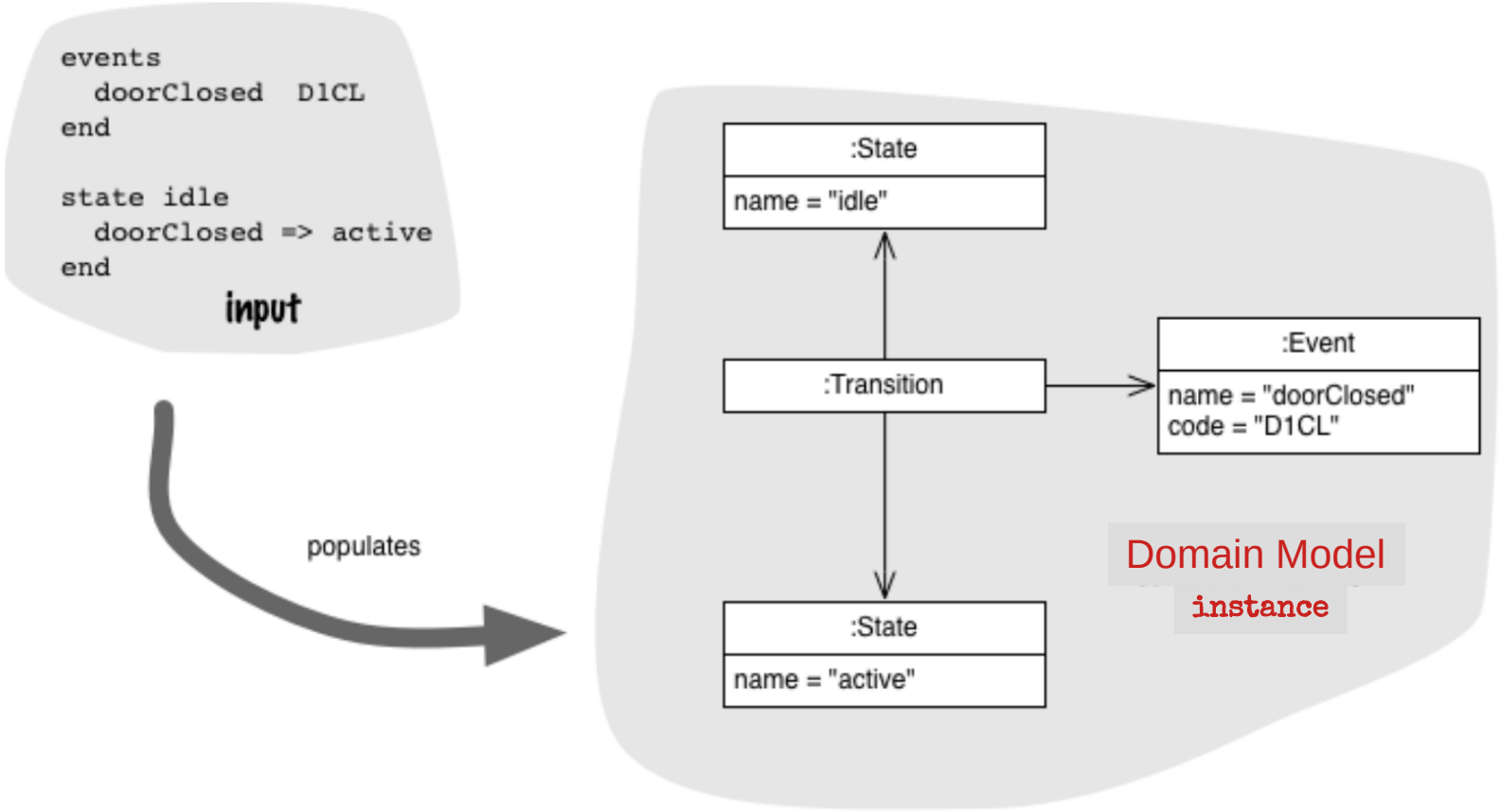
If the domain model is equipped with an interpreter or a way to be executed, then it acts as a semantic model since it defines the meaning of the DSL in terms of behavior.

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.

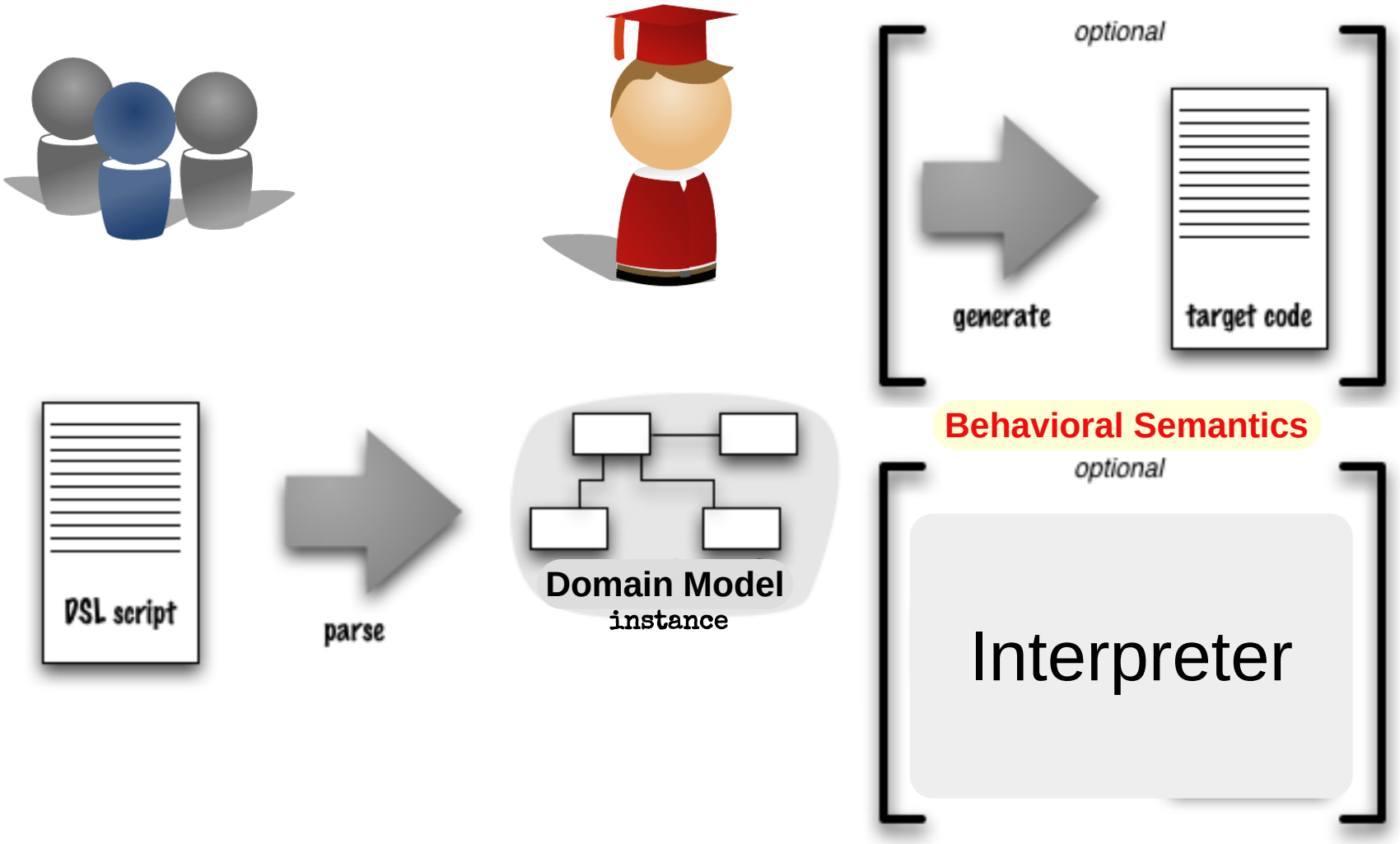
Semantic (or Domain) model



[Domain Specific Languages]

Modified

Semantic or Domain model ?



Modified

Modéliser un langage

-
-

syntax

```
while (b)
do
    C ;
done
```

semantics

Exécuter C de manière répétée (et sequentielle), aussi longtemps que l'expression *b* est vraie.

Modéliser un langage

-
-

syntax

```
while (b)
do
    C ;
done
```

semantics

Exécuter C de manière répétée (et sequentielle), aussi longtemps que l'expression *b* est vraie.

- 1) évaluer l'expression *b.*
  - si *b* == vrai, exécuter C et retourner à 1)
  - si *b* == faux, sortir.

Syntaxe et Sémantique comportementale

-
-

syntax

```
while (b && c)
do
    C ;
done
```

semantics

Exécuter C de manière répétée (et sequentielle), aussi longtemps que l'expression *b* && c est vraie.

- 1) évaluer l'expression *b.*
  - si *b* == vrai, exécuter C et retourner à 1)
  - si *b* == faux, sortir.

Ordre d'évaluation de b et c ?
Laziness ?

Sémantique comportementale

Axiomatic semantics

Hoare Triples

- Meaning of construct S can be described in terms of triples:

  { P } S { Q }

  - P and Q are formulas or assertions.
    - P is a precondition on S
    - Q is a postcondition on S
  - Asserts a fact (may be either true or false)
  - The triple is valid if:
    - execution of S begins in a state satisfying P
    - S terminates
    - resulting state satisfies Q

http://www.cs.purdue.edu/homes/suresh/565-Spring2009/lectures/lecture-6.pdf

Axiomatic semantics

-

while (b)
do
    C ;
done

```
while(x <= 10)
{
    x++;
}
```

{x ∈ ℤ} **while** B **do** C **od** { x ∈ ℤ ∧ x > 10}

Operational semantics

-
-

Condition

Rewriting rule

$\langle n, \sigma \rangle \Downarrow n$ "n in state σ, evaluates to n"

$\langle a, \sigma \rangle \Downarrow n$ "expression a in state σ, evaluates to n"

$\langle X, \sigma \rangle \Downarrow \sigma(X)$ "location X evaluates to its contents in a state"

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \Downarrow \sigma} \quad \text{(while loops)}$$

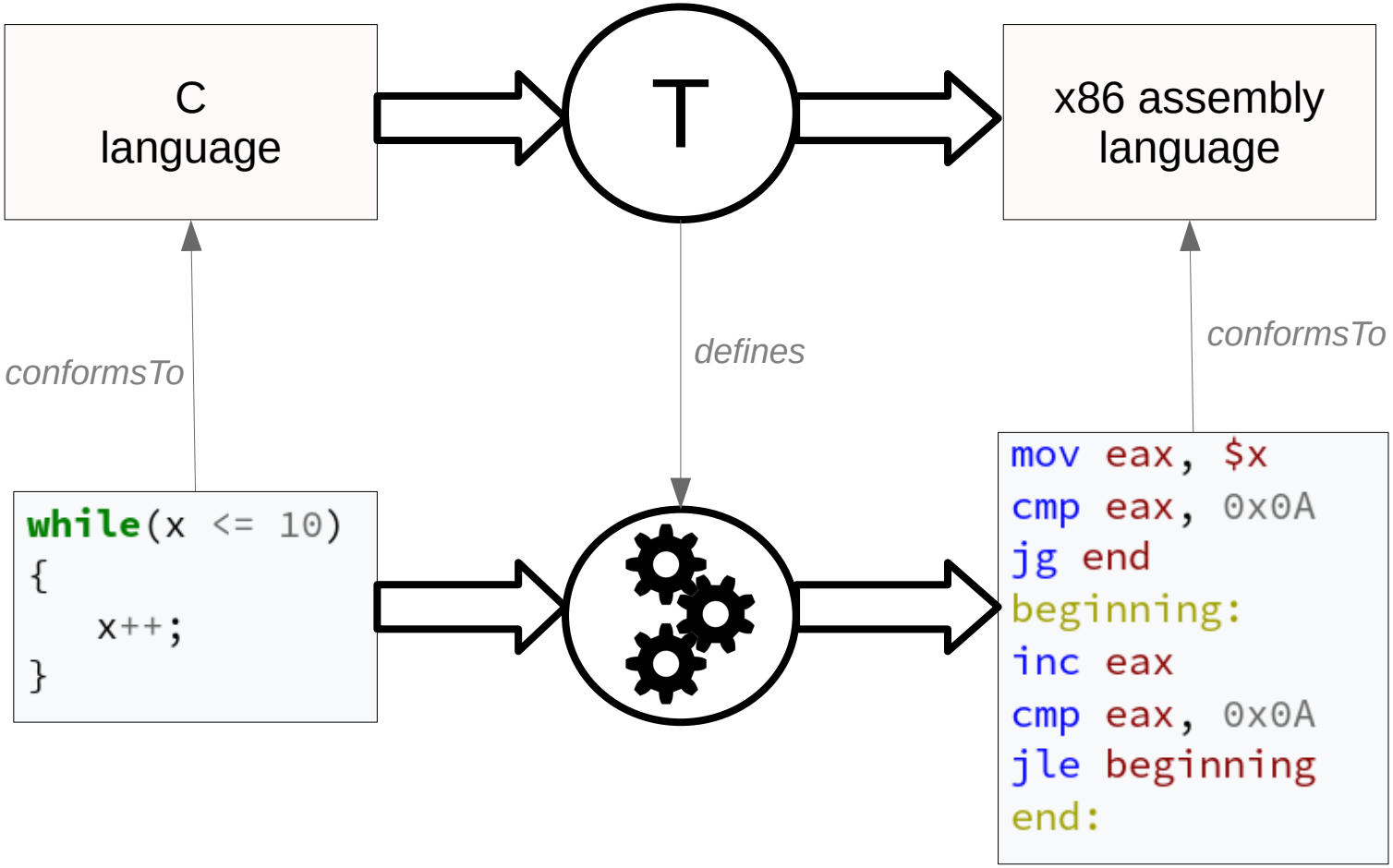
```
while (b)
do
    C ;
done
```

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma'' \quad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \Downarrow \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \Downarrow \sigma'}$$

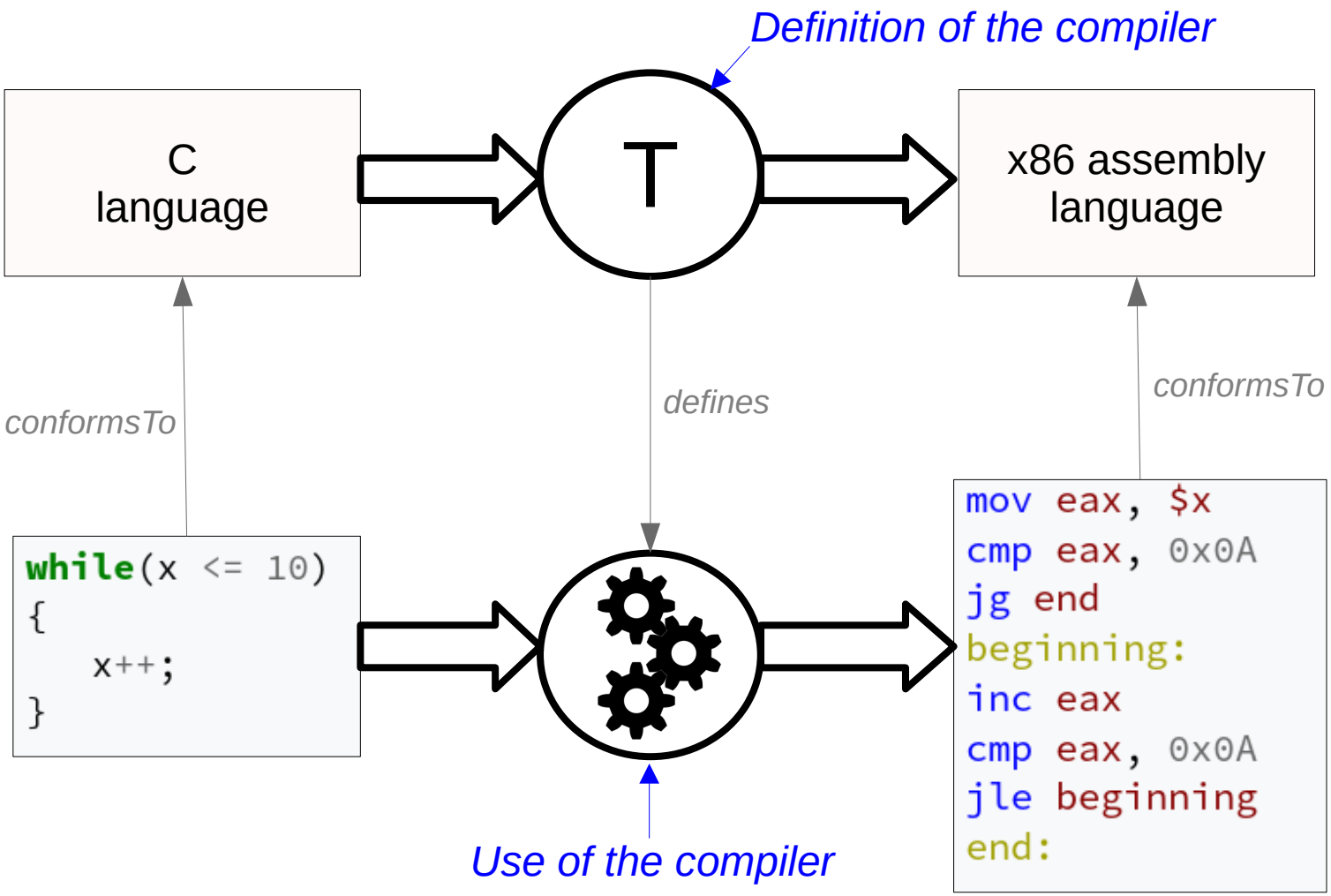
$$\frac{\langle B, s \rangle \Rightarrow \mathbf{true}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s \rangle \longrightarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, s' \rangle} \qquad \frac{\langle B, s \rangle \Rightarrow \mathbf{false}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s \rangle \longrightarrow s}$$

From wikipedia

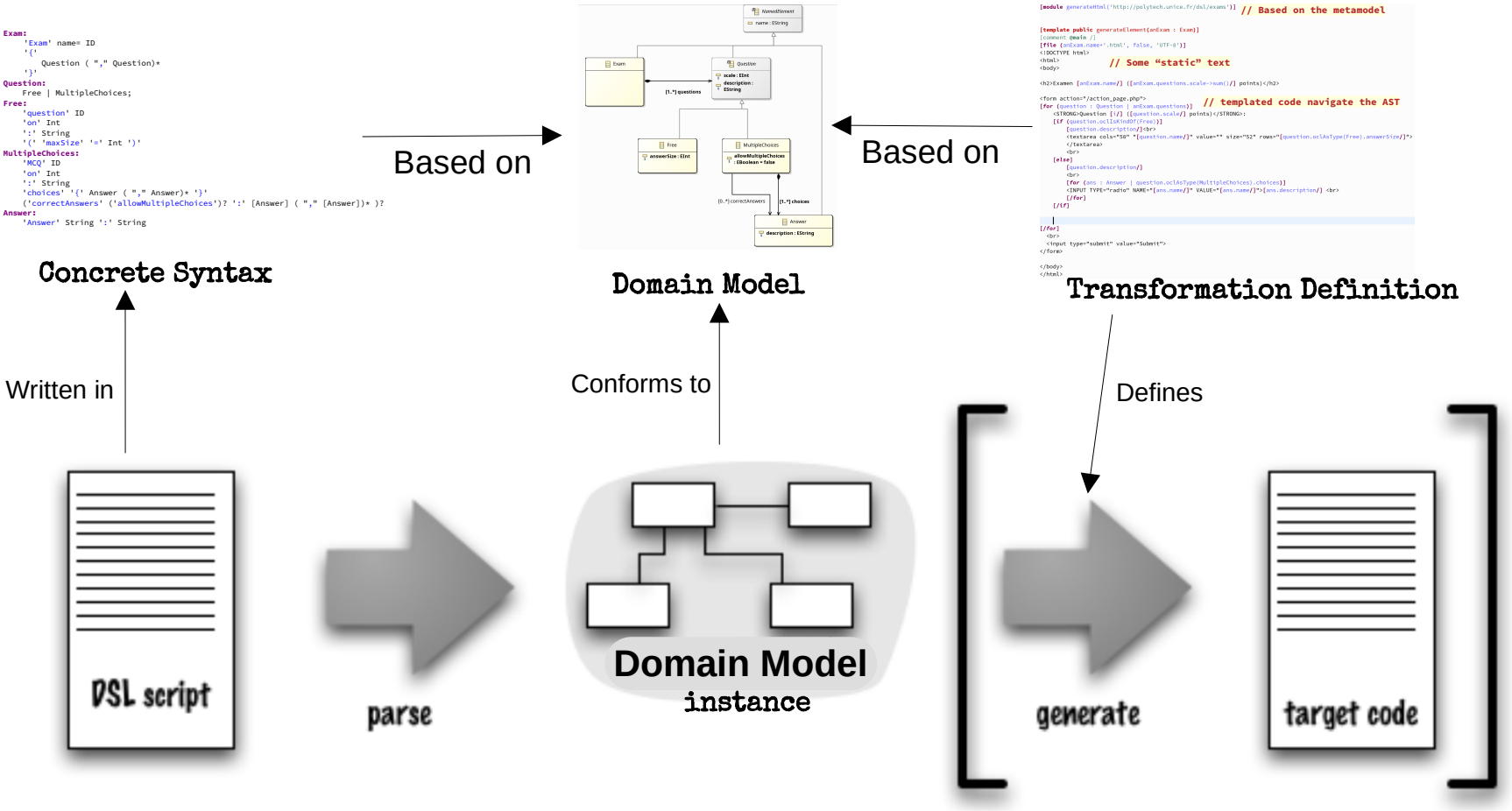
Transformational semantics



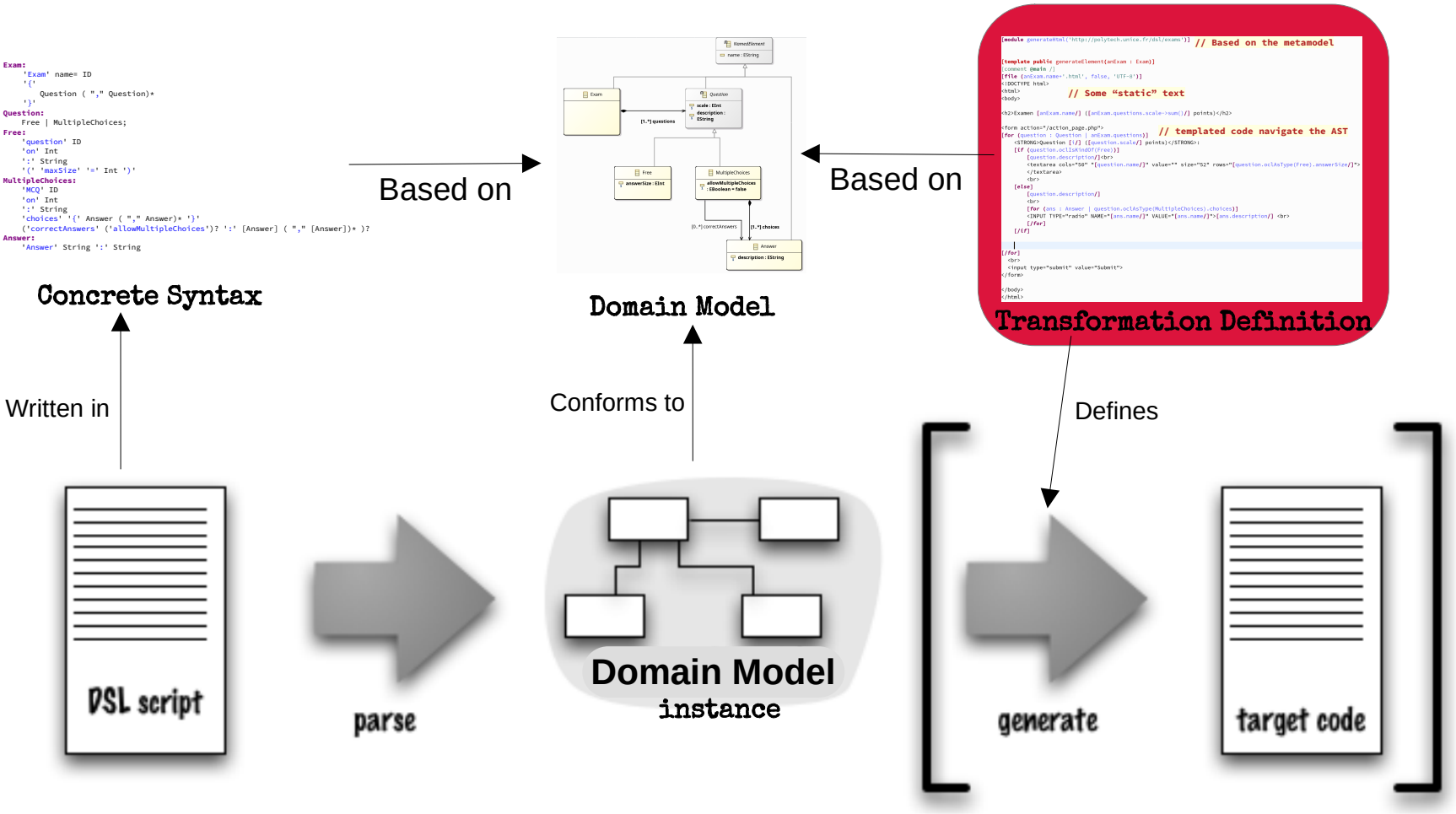
Transformational semantics



Languages artifacts



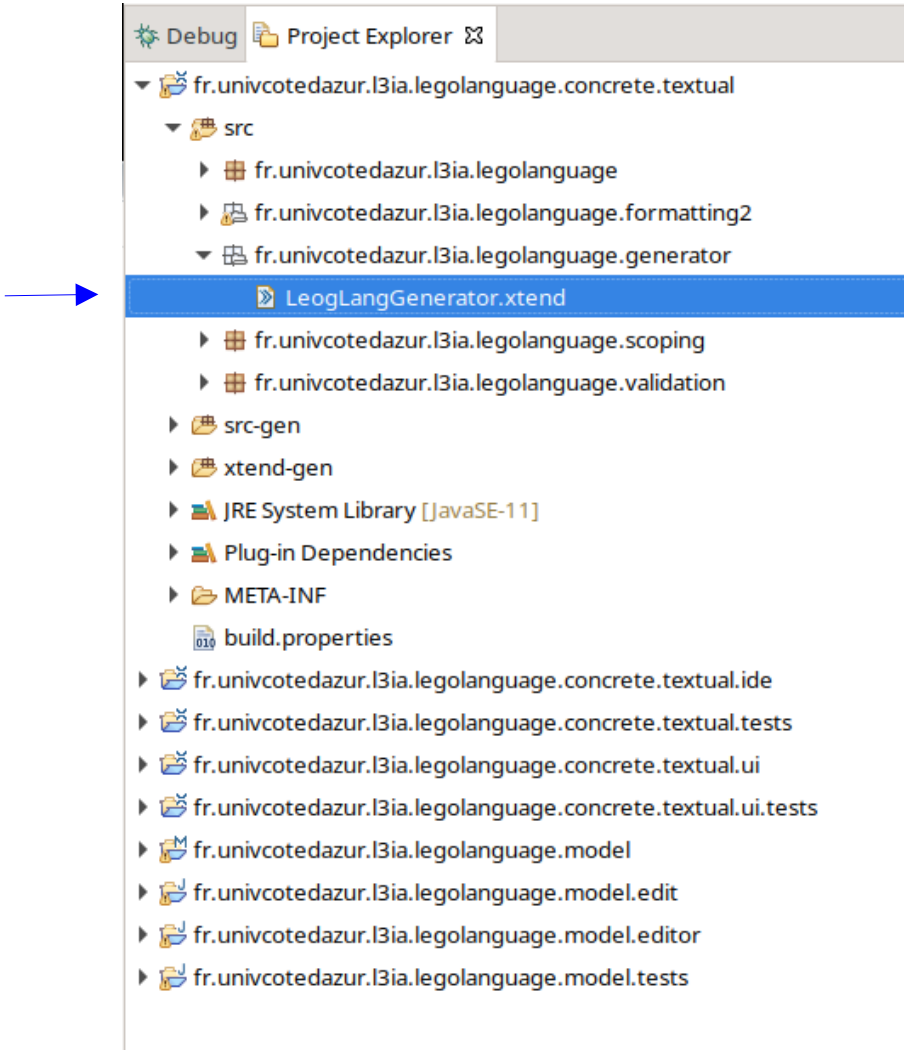
Languages artifacts



Transformation definition

-
-
-
-
-

Transformation definition



Transformation definition

```
/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LegoLangGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        var LegoProgram prog = resource.allContents.head as LegoProgram
        var String fileContent = ''
        fileContent += RobotToString(prog.robot)
        for (v : prog.ownedVariables){
            fileContent += VarToString(v)
        }
        for (s : prog.ownedStatements){
            fileContent += StatementToString(s)
        }
        fsa.generateFile(prog.name+'.py', '#!/usr/bin/env python3

# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *\n\n' + fileContent)
    }
```

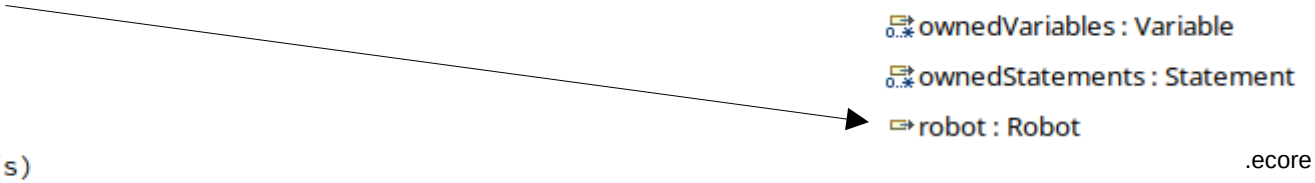
Target language is Python3, and we always use specific librairies for the lego robot

Transformation definition

```
/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LegoLangGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        var LegoProgram prog = resource.allContents.head as LegoProgram
        var String fileContent = ''
        fileContent += RobotToString(prog.robot)
        for (v : prog.ownedVariables){
            fileContent += VarToString(v)
        }
        for (s : prog.ownedStatements){
            fileContent += StatementToString(s)
        }
        fsa.generateFile(prog.name+'.py', '#!/usr/bin/env python3

# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *\n\n' + fileContent)
    }
```



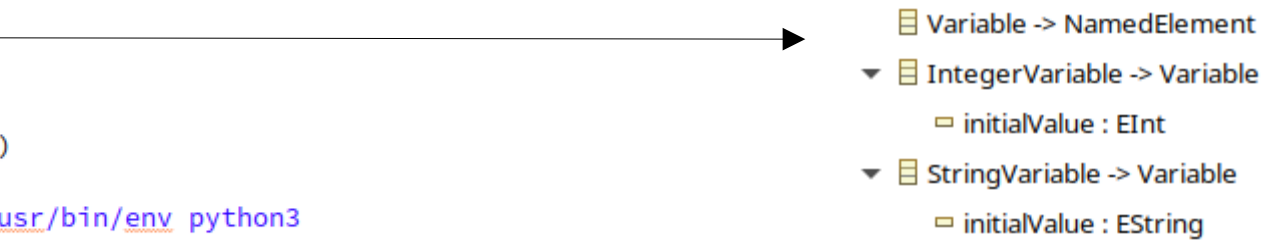
.ecore

Transformation definition

```
/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LegoLangGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        var LegoProgram prog = resource.allContents.head as LegoProgram
        var String fileContent = ''
        fileContent += RobotToString(prog.robot)
        for (v : prog.ownedVariables){
            fileContent += VarToString(v)
        }
        for (s : prog.ownedStatements){
            fileContent += StatementToString(s)
        }
        fsa.generateFile(prog.name+'.py', '#!/usr/bin/env python3

# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *\n\n' + fileContent)
    }
```

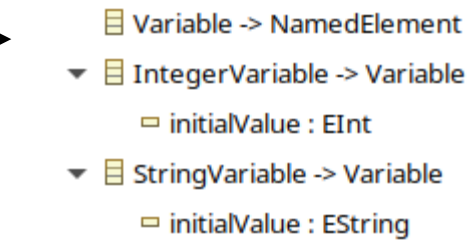


.ecore

Transformation definition

```
/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LegoLangGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        var LegoProgram prog = resource.allContents.head as LegoProgram
        var String fileContent = ''
        fileContent += RobotToString(prog.robot)
        for (v : prog.ownedVariables){
            fileContent += VarToString(v)
        }
        for (s : prog.ownedStatements){
            fileContent += StatementToString(s)
        }
        fsa.generateFile(prog.name+'.py', '#!/usr/bin/env python3
```



.ecore

```
# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *\n\n' + fileContent)
    }
```

```
def String VarToString(Variable v) {
    var res =''
        if (v instanceof IntegerVariable){
            res+=v.name+':int = '+v.initialValue+'\n'
        }else
        if (v instanceof StringVariable){
            res+=v.name+':str = \''+v.initialValue+'\'\n'
        }
    return res
}
```

Transformation definition

```
/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LegoLangGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        var LegoProgram prog = resource.allContents.head as LegoProgram
        var String fileContent = ''
        fileContent += RobotToString(prog.robot)
        for (v : prog.ownedVariables){
            fileContent += VarToString(v)
        }
        for (s : prog.ownedStatements){
            fileContent += StatementToString(s)
        }
        fsa.generateFile(prog.name+'.py', '#!/usr/bin/env python3

# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *\n\n' + fileContent)
    }
```

```
def String StatementToString(Statement s) {
    var res =''
        if (s instanceof Addition){
            res+=AdditionToString(s as Addition)
        }else
        if (s instanceof Assignment){
            res+=AssignementToString(s as Assignment)
        }else
        if (s instanceof VariableRef){
            res+=VariableRefToString(s as VariableRef)
        }else
        if (s instanceof IntegerLiteral){
            res+=(s as IntegerLiteral).value
        }else
        if (s instanceof StringLiteral){
            res+=(s as StringLiteral).value
        }else
        if (s instanceof Start){
            res+=StartToString(s as Start)
        }
    return res
}
```

Transformation definition

test1.leg

```
LegoProgram test1
Robot r1 {
    sensor{
        LightSensor ls1 on 1
    }
    actuator{
        Motor left on A
        Motor right on B
    }
}

int i = 10
int j = 20

j := ( i + j )
start motor "r1.left" at j
start motor "r1.right" at (i + 10)

string s  ="toto"
```

```
#!/usr/bin/env python3

# Import the necessary libraries
import time
import math
from ev3dev2.motor import *
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor.virtual import *



#robot sensor/actuator part
ls1= ColorSensor(INPUT_1)
left= LargeMotor(OUTPUT_A)
right= LargeMotor(OUTPUT_B)


#variable declaration part
i:int = 10
j:int = 20
s:str = 'toto'


#Statements start
j = (i + j)
left.on(j)
right.on((i + 10))
```