Model-Based Software Engineering

Lecture 07 – Introduction to Semantics, Transformation, Execution, Analysis

Prof. Dr. Joel Greenyer

May 31, 2016

Acknowledgment

- The slides of this lecture are inspired by lecture slides from
  - *Ekkart Kindler*: Course on Advanced Topics in Software Engineering, DTU Compute, 2015.
    - http://www2.imm.dtu.dk/courses/02265/f15/schedule.shtml
  - *Ina Schäfer, Christoph Seidl*: Modellbasierte Software-entwicklung, TU Braunschweig, 2015.
  - *Steffen Becker*: Model-Driven Software Development, Universität Paderborn, 2013
  - The Eclipse Open Model CourseWare (OMCW) Project:
    - https://eclipse.org/gmt/omcw/

SOFTWARE ENGINEERING

5.1. Introduction to semantics, transformations, execution, analysis

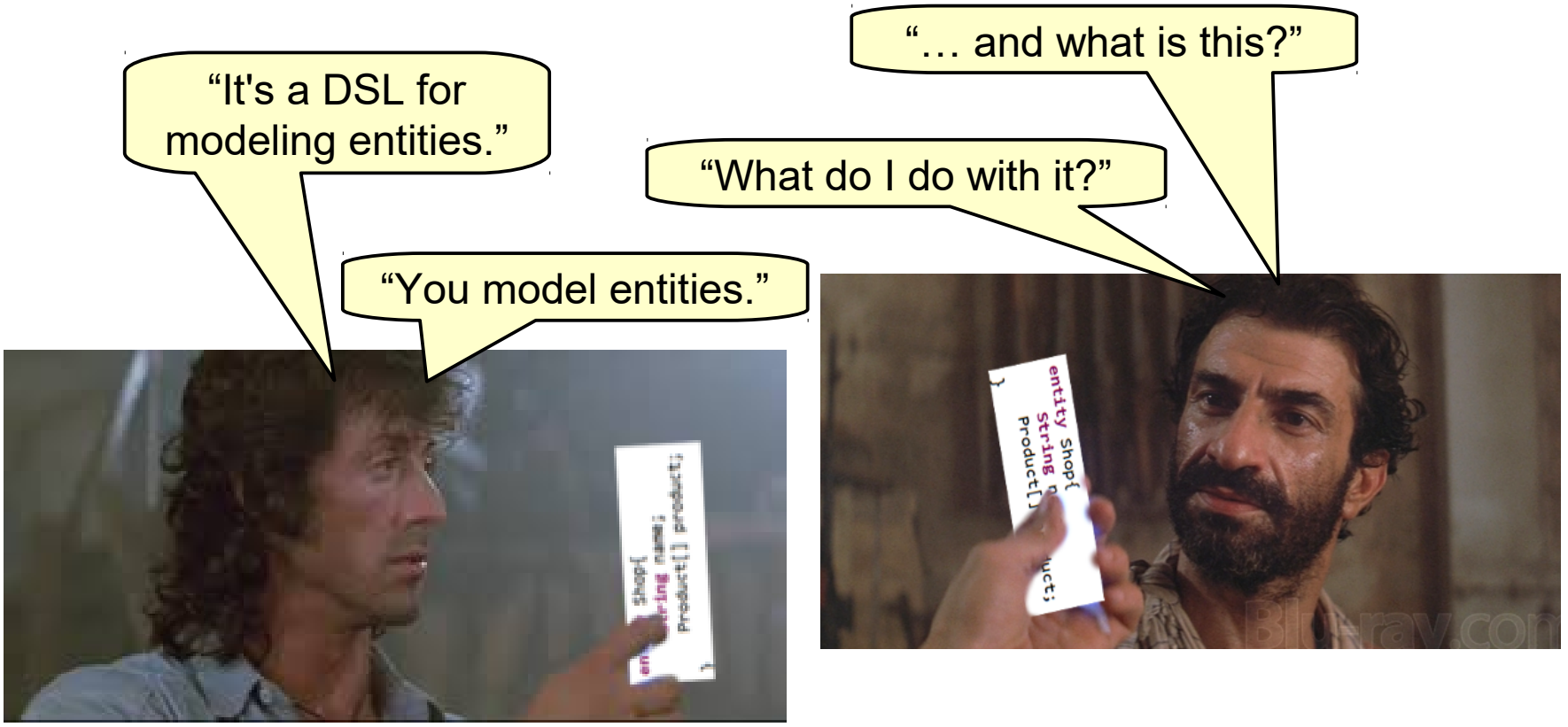
Leibniz Universität Hannover

Previously

- In the previous lectures, we covered several aspects of defining a formal language (possibly domain specific)
  - Metamodeling
    - including OCL
  - Defining a concrete syntax
    - graphical
    - textual

Formal Language

- So now you have a formal language...
- ...what do I do with it?



Formal Language

- So now you have a formal language...
- ...what do I do with it?
  - Code generation and **execution**
    - maybe for multiple platforms
  - Code generation for **generating other software artifacts**
    - tests, documentation, database configuration, ...
  - Analysis
    - performance, dependability, formal checks, …
- In order to do these things, we need to define the meaning of our formal language---its **semantics**

Formal Language

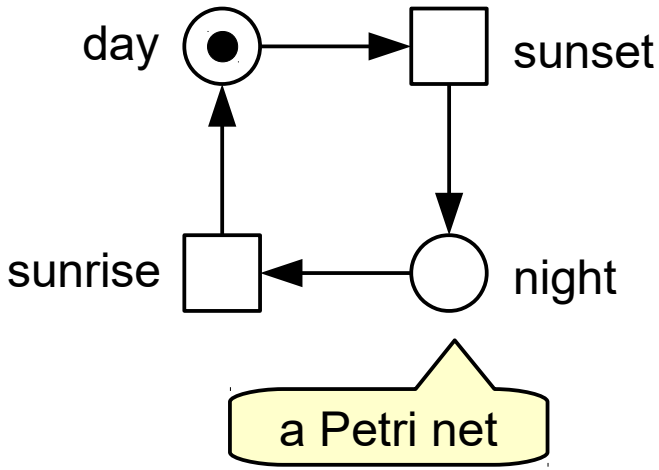
- (see Lecture 2)

- A **formal language definition** contains the definition of
  - the **abstract syntax**: defines its internal structure
    - Defines the language constructs and how they can be combined
  - the **concrete syntax**: defines its **notation,** its visual representation for the user (textual or graphical)
  - the **semantics**: defines the meaning of the language constructs and their combinations
  - (sometimes also) the **serialization syntax**: how are sentences of the language stored or exchanged by tools

What is the Semantics of these Models?

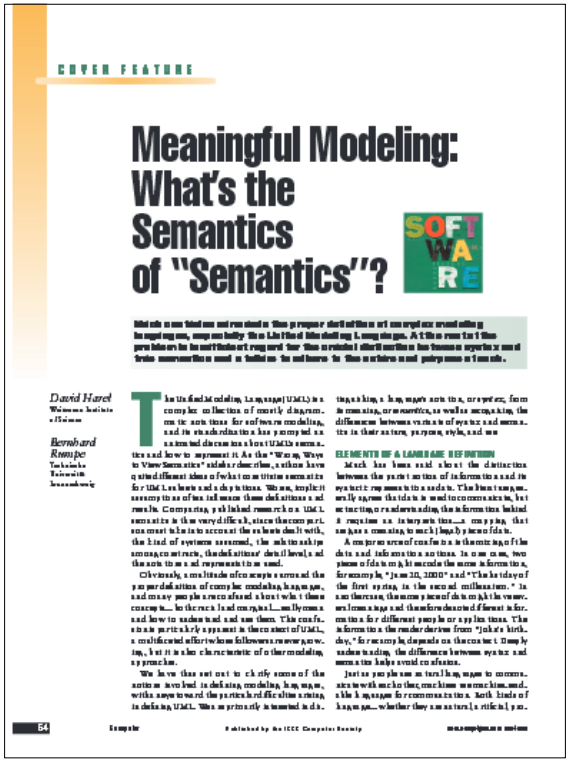
```
entity Shop{
        String name;
        Product[] product;
}

entity Product{
        String name;
        Int productNumber;
}

entity Book extends Product{
        String ISBN;
        String publisher;
}

entity CD extends Product{
        String label;
}
```

an 'Entities' model



What's the Semantics of Semantics?

- D. Harel and B. Rumpe, "Meaningful modeling: what's the semantics of "semantics"?," in Computer, vol. 37, no. 10, pp. 64-72, Oct. 2004.
  - http://www.wisdom.weizmann.ac.il/~harel/papers/ModSemantics.pdf



COVER FEATURE

Meaningful Modeling: What's the Semantics of "Semantics"?

SOFTWARE

David Harel
Weizmann Institute of Science

Bernhard Rumpe
Technische Universität Braunschweig

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is the metamodel":**
  - Metamodels describes the language's syntax
  - required to describe the semantics, but not semantics itself

- **"Semantics is the semantic domain":**
  - Example: "The semantics of the Entities language is UML class diagrams"
  - *Semantic Domain*: Another language or logic that can be used to describe the semantics of a language
  - But to name a semantic domain alone is not enough
  - it is required to describe a **detailed mapping** of how the constructs of your language correspond to concepts in the semantic domain

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is the context conditions":**
  - In compiler theory, the term "semantic analysis" refers to a process after parsing (which is usually based on a context-free grammar) that further extends the AST based on further rules that could not be expressed in the grammar
    - for example for variable scoping
  - Similarly, some refer to, for example, additional OCL constraints as "semantics"
    - for example, rules for valid arcs in the Petri net metamodel
  - But this actually **only refines the syntax definition**

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is the context conditions":**
  - In compiler theory, the term "semantic analysis" refers to a process after parsing (which is usually based on a context-free grammar) that further extends the AST based on further rules that could not be expressed in the grammar
    - for example for variable scoping
  - Similarly, some refer to, for example, additional OCL constraints as "semantics"
    - for example, rules for valid arcs in the Petri net metamodel
  - But this actually **only refines the syntax definition**

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is dealing with behavior":**
  - Many formal languages deal with behavior
    - programming languages, Petri nets, Statecharts, …
  - Their semantics definitions must describe the behavior for each valid program/model
  - But not all formal languages describe behavior
  - Many describe structures (for example the Entities language), mathematical expressions, database queries, etc.
  - Their semantics does not relate to the behavior of a system
  - Thus, semantics and behavior should not be confused

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is being executable":**
  - "If a language is executable it has a well-defined semantic"
    - probably true
    - but it may not be given in a clear representation
      - for example: semantics could be given only in the form of a code generator implementation, which is hard to understand
    - also, executable languages with unclear semantics exist
  - "If a language has a semantic, it is executable"
    - not always true
    - again, some languages do not deal with behavior
    - not all behavior semantics are executable semantics
      - example (relates, but subtle difference):
        - » One semantics of finite state automata describes all possible sequences in which transitions can fire (all possible executions)
        - » Another semantics defines which transition fires next in a given current state, and what the current state will be in the next step

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

- **"Semantics is the meaning of individual constructs":**
  - "People often refer to the semantics of some part of the language, even just one construct. Clearly, there is much more to semantics than that."

- **"Semantics means looking mathematical"**
  - "When some people see that parts of a language definition have mathematical symbols, they are convinced that it is probably also precisely defined. This is simply not true."

Wrong Ways to View Semantics

(from "Meaningful modeling: what's the semantics of "semantics"?")

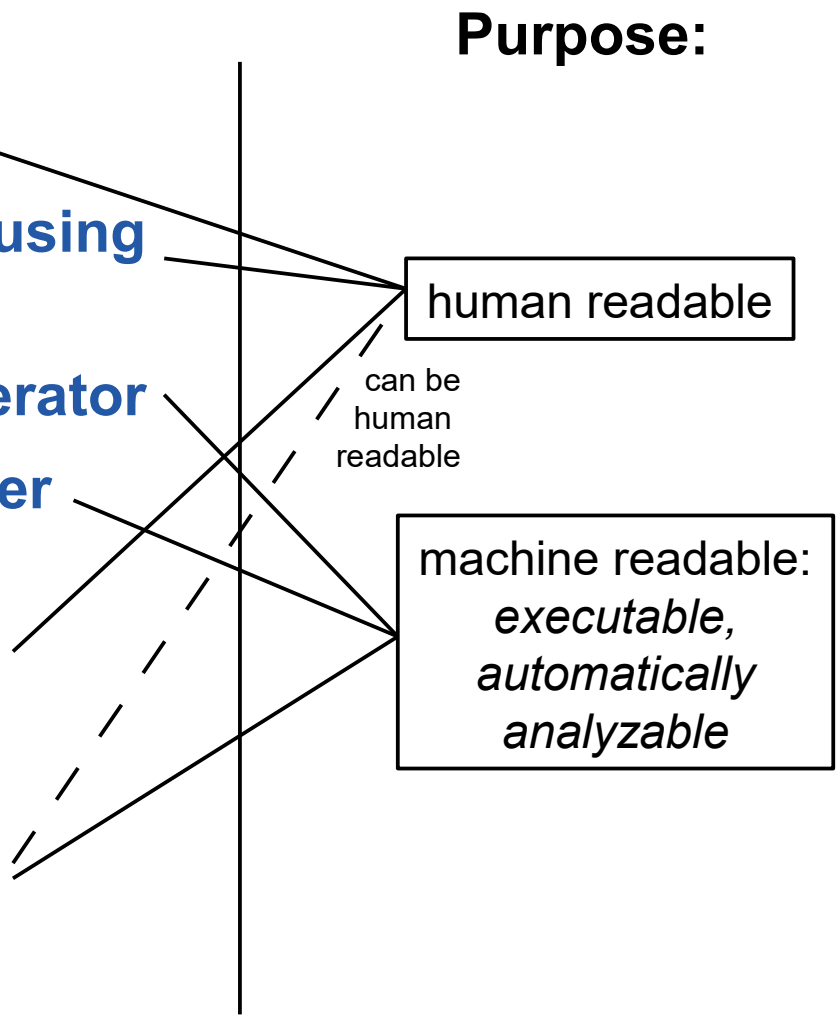
- **"Semantics is _____"**
  - Some people simply give a buzzword to indicate something about how the semantic definition goes, as in "the semantics is given by message-passing". This prompts others to think that the language is properly endowed with semantics. Sadly, the worst cases are when the people making this kind of statement actually believe it themselves."

Ways to Define Semantics?

- How do we define the semantics of a formal language?
  - By **using natural language**
  - By **giving a formal definition using mathematics**
  - By **implementing a code generator**
    - for example: Generate Java code from state machines
  - By **implementing an interpreter ("virtual machine")**
  - By **specifying a mapping to a semantic domain model**
    - for example: Mapping Activity diagrams to Petri nets
  - By **implementing a model transformation to a semantic domain model**
    - same as above, but executable
    - A code generator can also be seen as a model transformation

Ways to Define Semantics?

- How do we define the semantics of a formal language?
  - By **using natural language**
  - By **giving a formal definition using mathematics**
  - By **implementing a code generator**
  - By **implementing an interpreter ("virtual machine")**
  - By **specifying a mapping to a semantic domain model**
  - By **implementing a model transformation to a semantic domain model**



Giving a Formal Semantics Definition using Mathematics

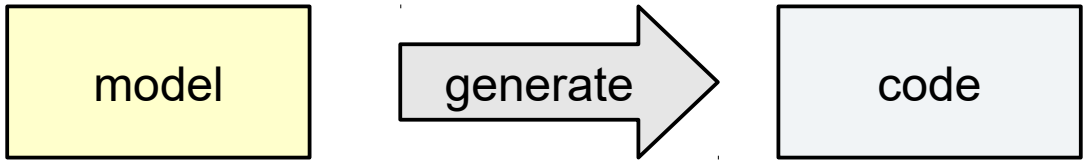
- for example: giving an operational semantics with rules that define the changes of the program state by an if-then-else statement in a simple programming language:

$$\frac{\langle b, \sigma\rangle \to \mathit{true} \quad \langle c_0, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'} \qquad \frac{\langle b, \sigma\rangle \to \mathit{false} \quad \langle c_1, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}$$

(first rule: “if b is true in state σ, and executing statement c_0 in state σ leads to state σ', then '**if** b **then** c_0 **else** c_1' in state σ leads to state σ' ”)

- (Details of this method of semantics definitions will not be part of this lecture)

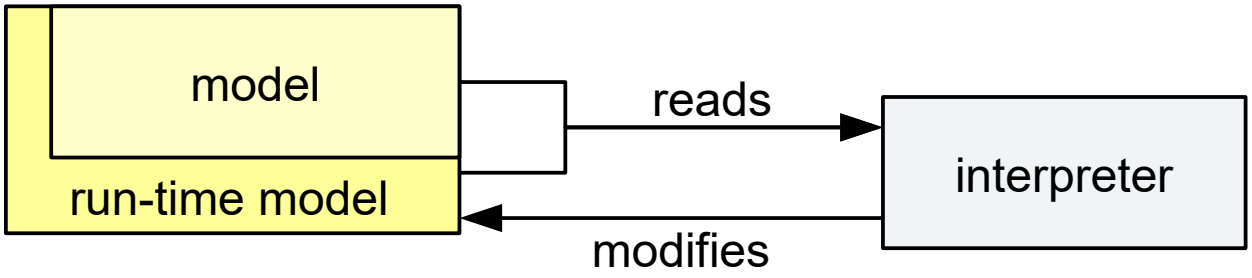
Code Generator



- for example: A state machine to Java generator defines the semantics of state machines **by a mapping to Java**
  - the semantics of Java is precisely specified in a specification
    https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf
  - the semantics of Java is also precisely defined **through its mapping to Java byte code**,
    - which is again precisely specified in a specification, see
      https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf
    - or for which the semantics is defined **in the form of different virtual machine implementations**

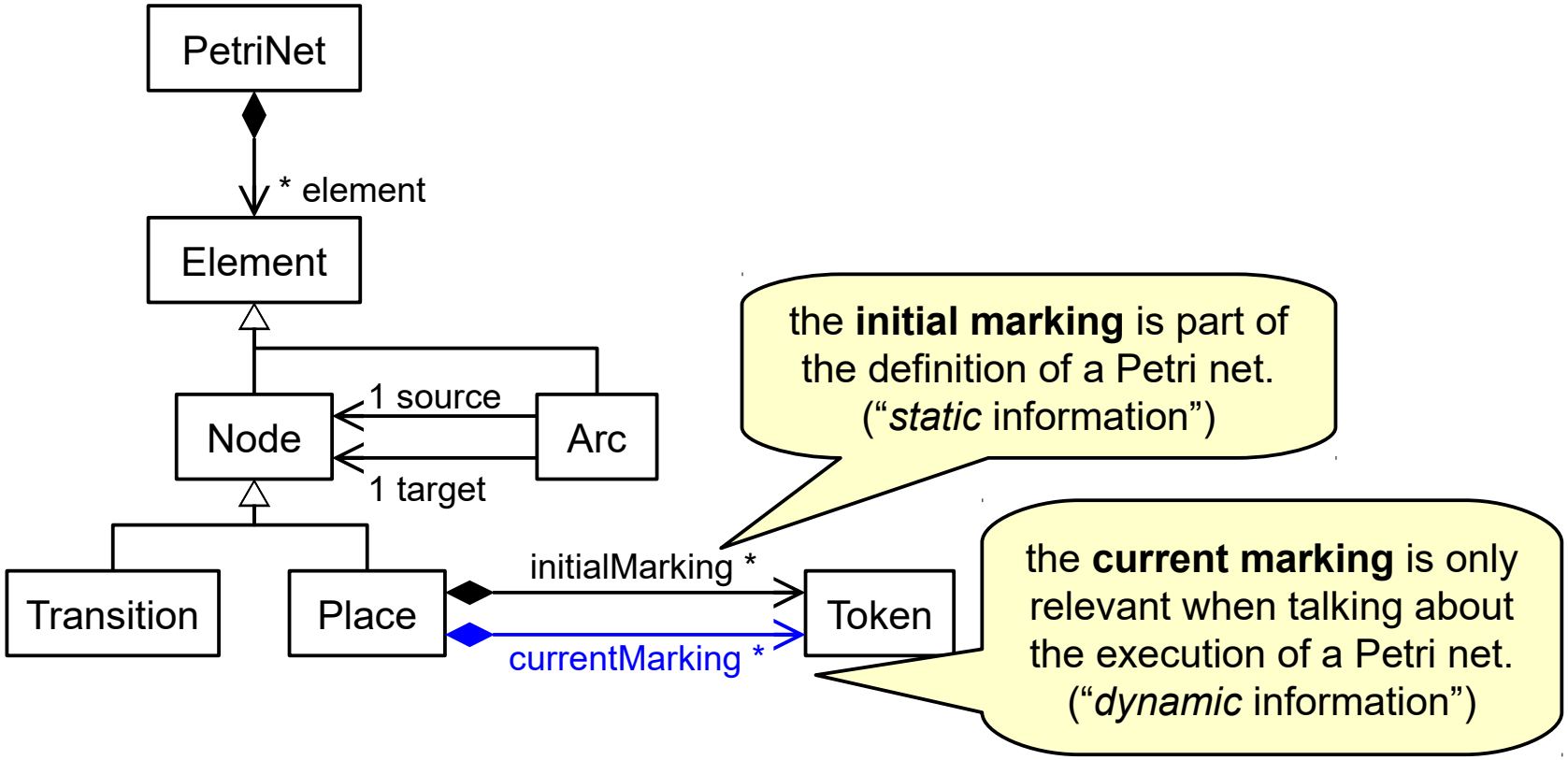
Programming an Interpreter ("Virtual Machine")

- for languages dealing with behavior, we can extend the metamodel by constructs that capture **run-time concepts**
  - for example: model "heap", "stack", "variable bindings", etc. for a programming language
- The interpreter can read the model and its runtime extension
- The runtime extension part captures the "current state" of execution, which the interpreter can modify



Interpreter and Run-time Model Extension

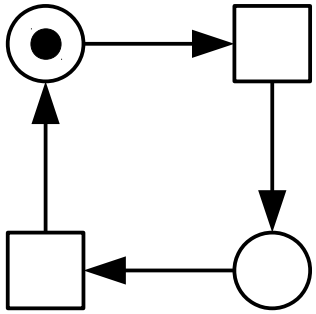
- Example: Petri net runtime extension



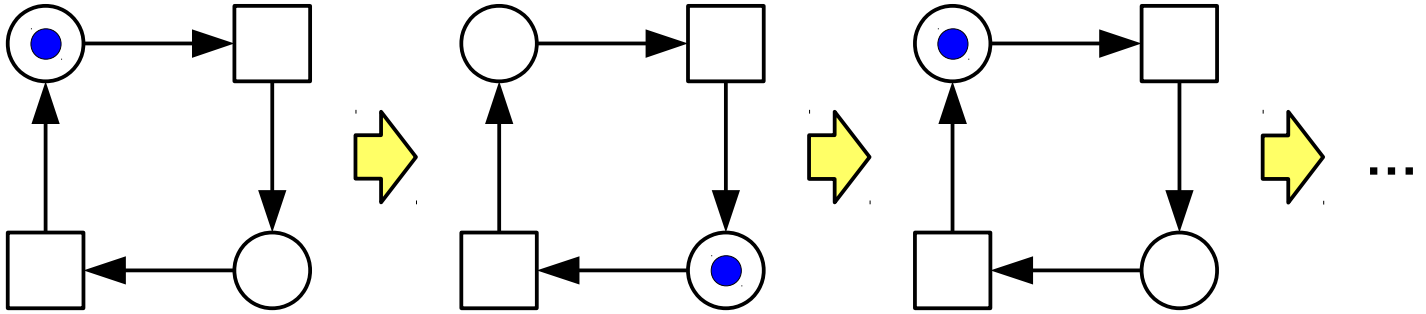
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension

static:

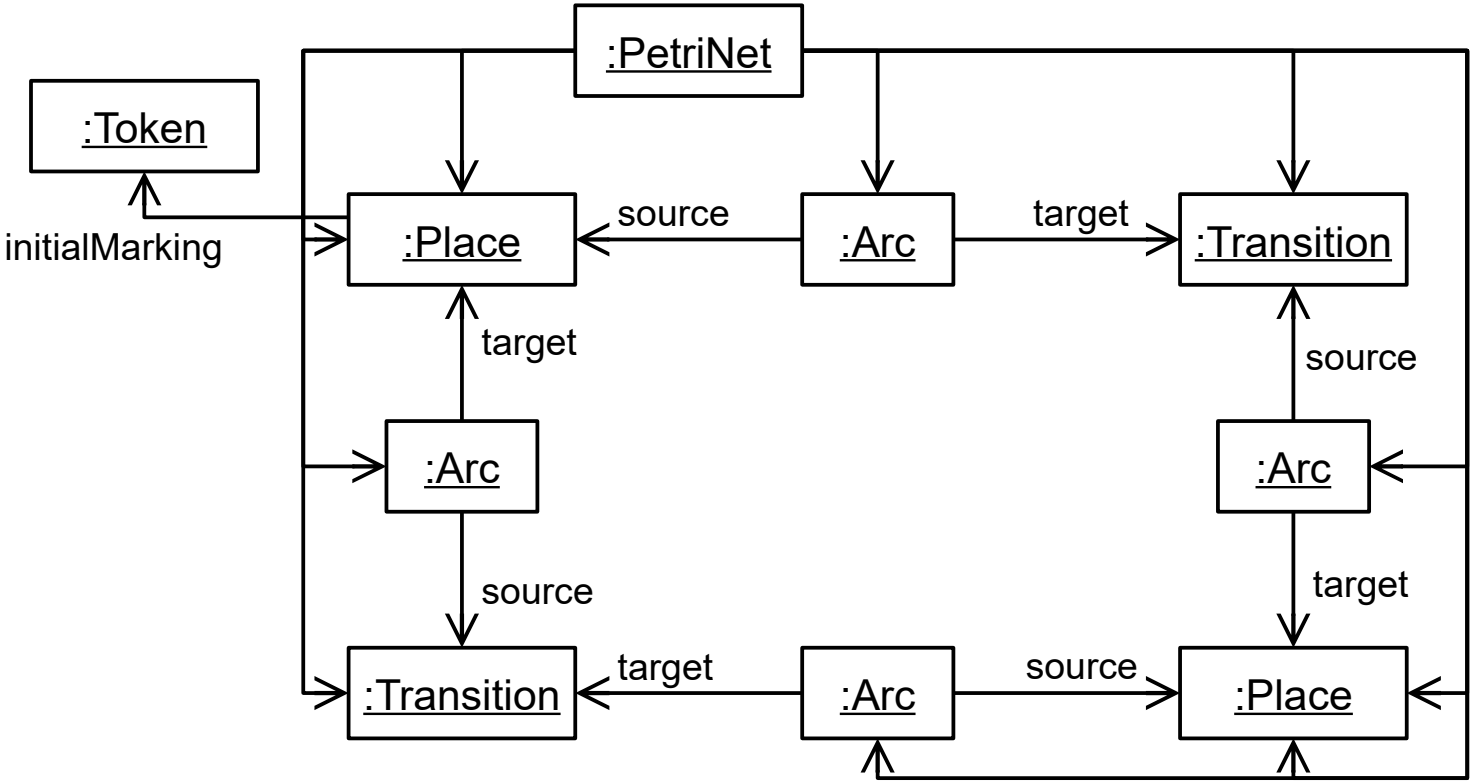


dynamic:



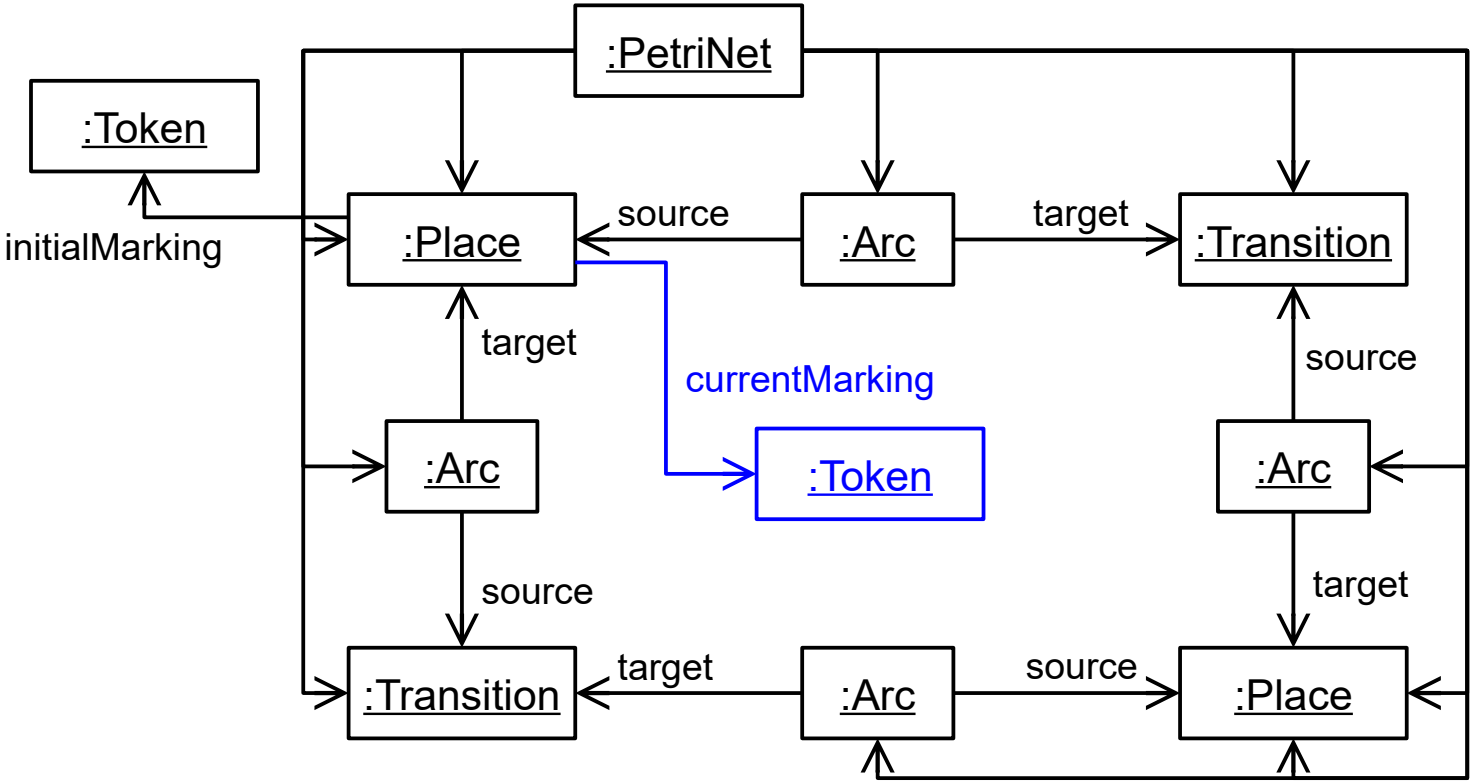
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension



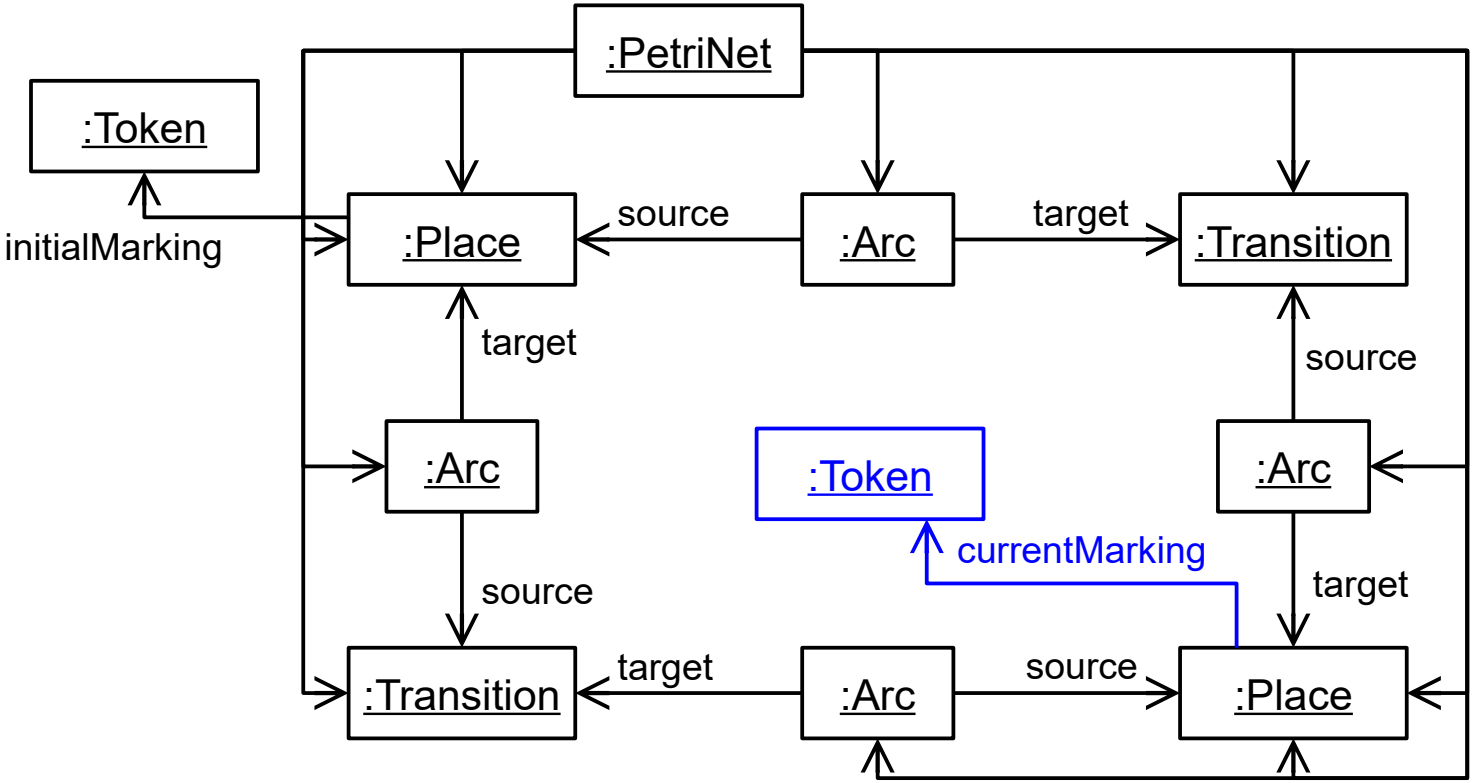
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension



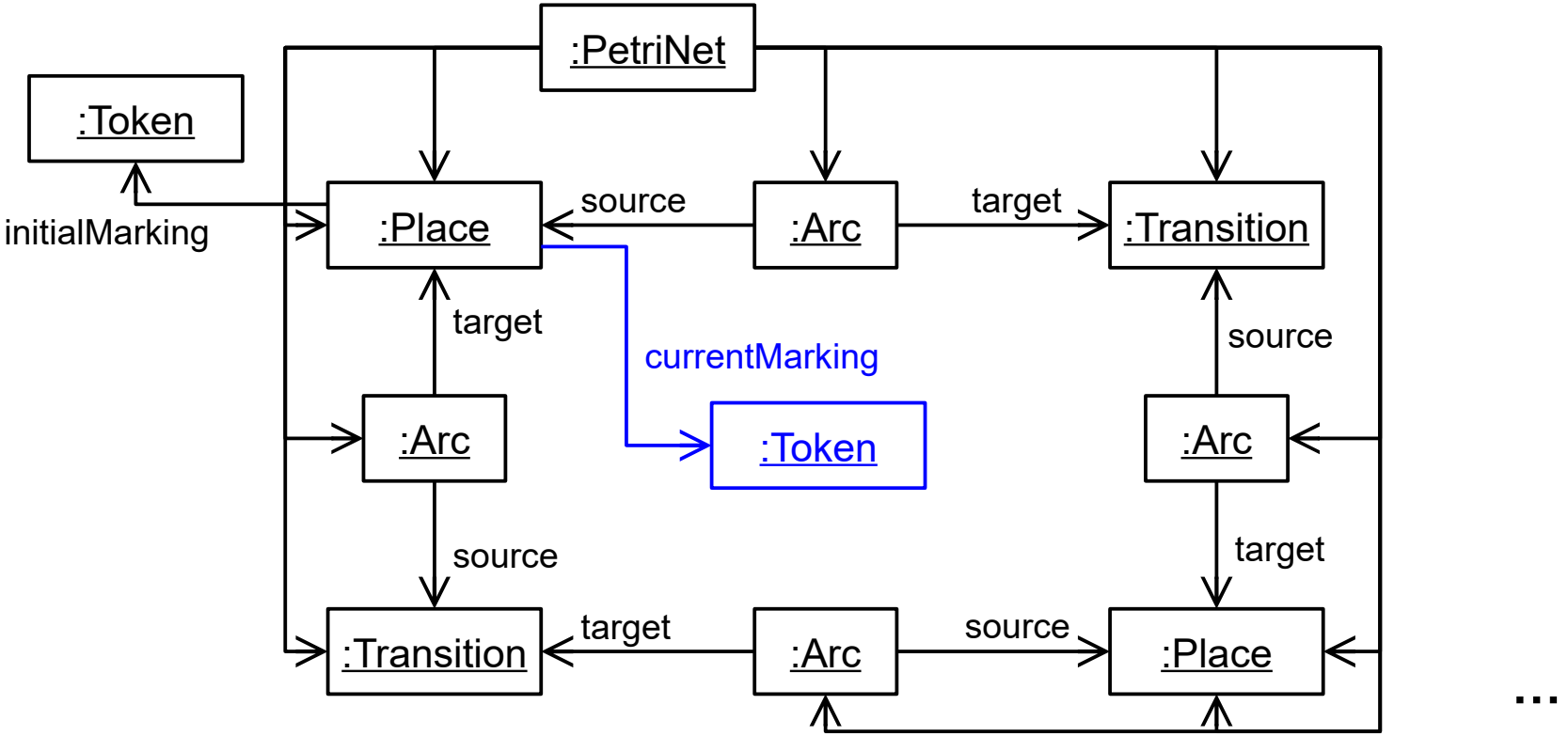
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension



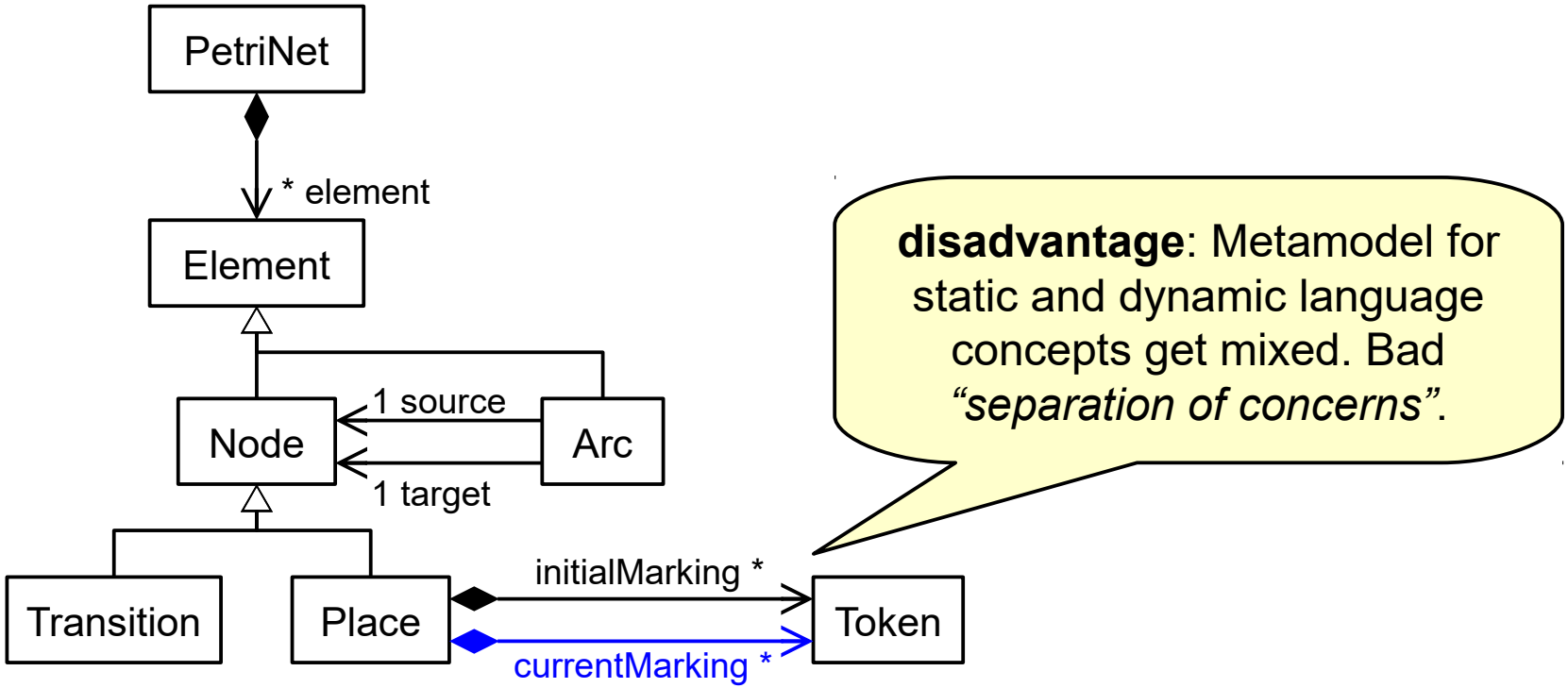
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension



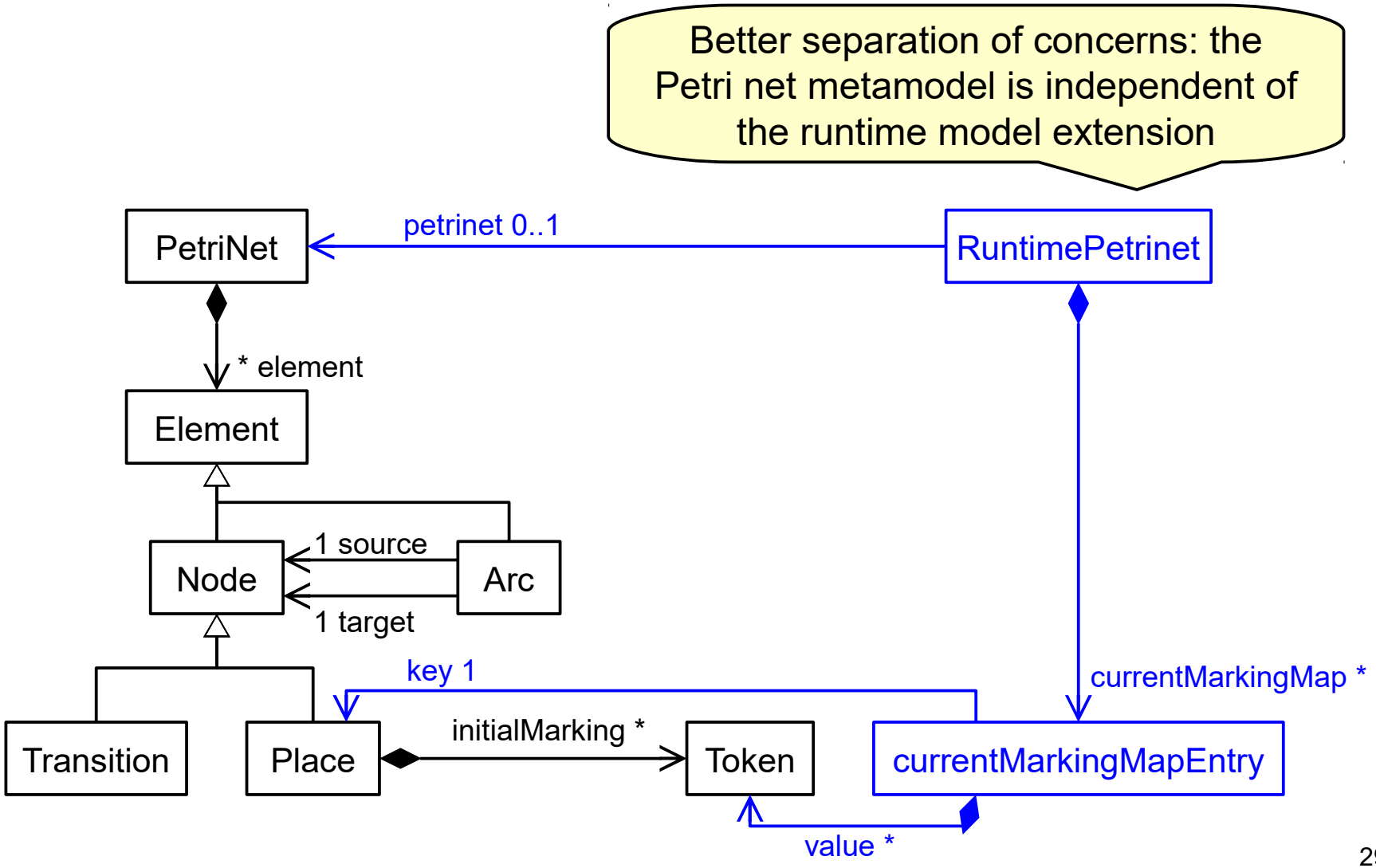
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension



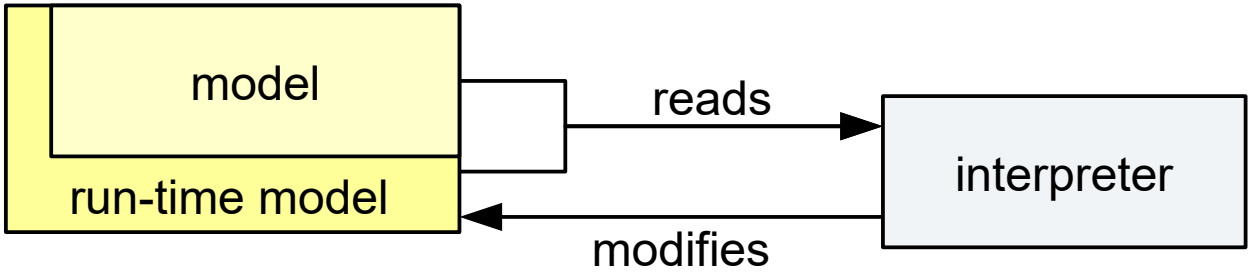
Interpreter and Run-time Model Extension

- Example: Petri net runtime extension

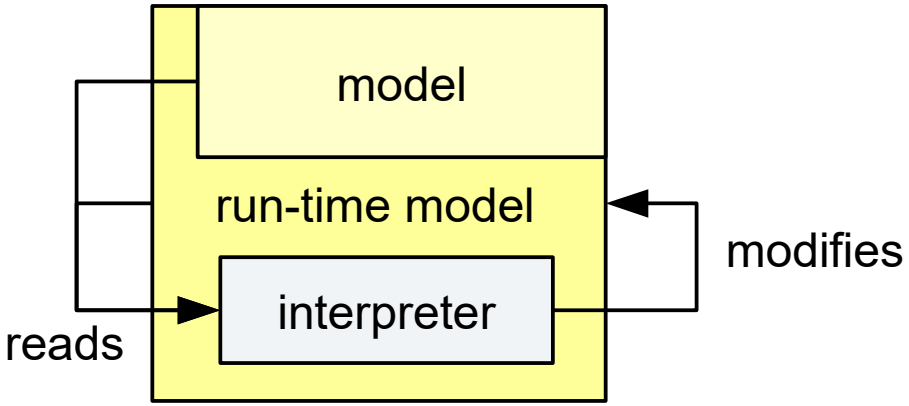


Interpreter and Run-time Model Extension

- The interpreter
  - can be an **external program** working on a Petri net model + runtime extension

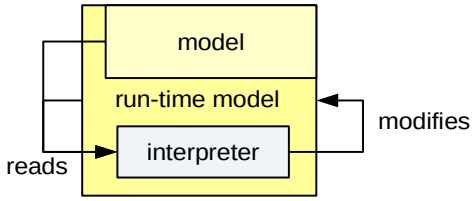


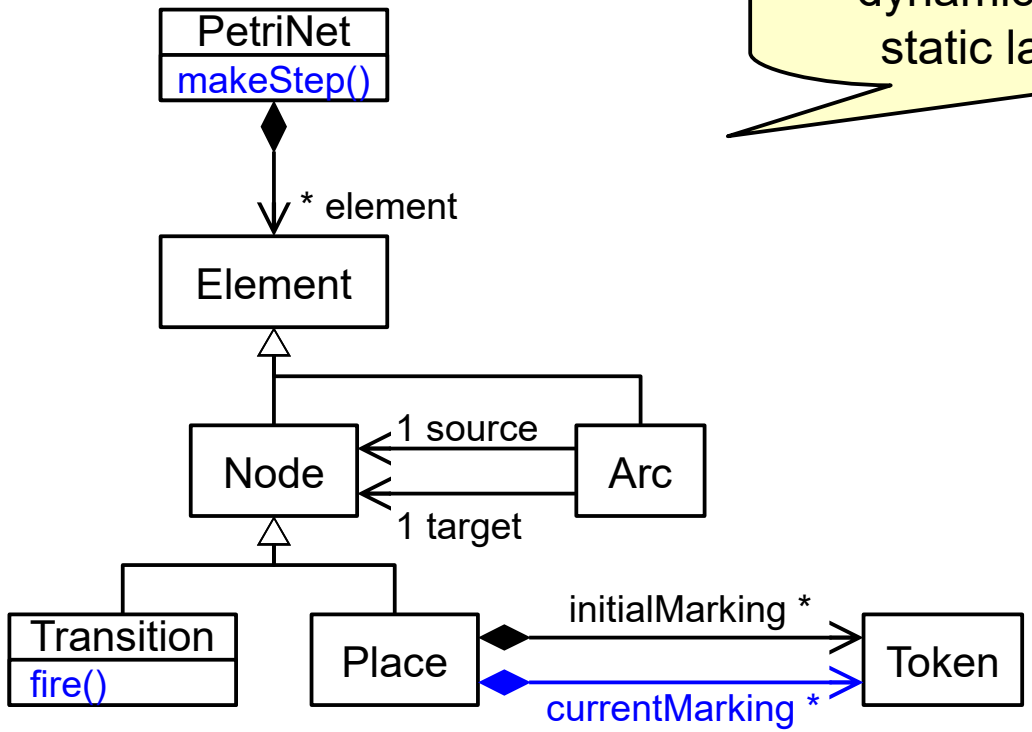
  - Or the interpreter can be part of the runtime model



Interpreter and Run-time Model Extension

- Example: Petri net interpreter part of the runtime model

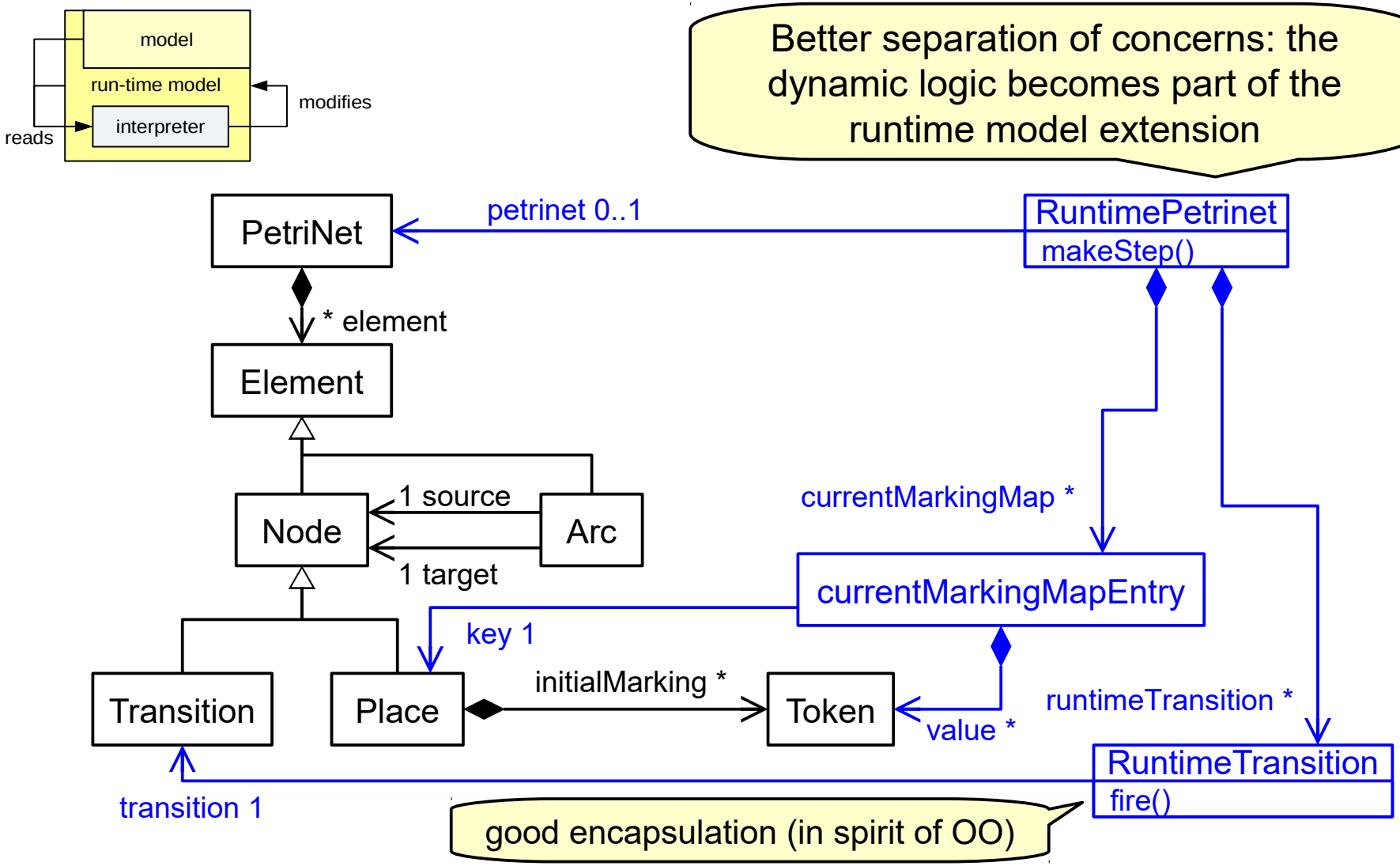




again, bad separation of concerns: dynamic logic becomes part of static language metamodel.

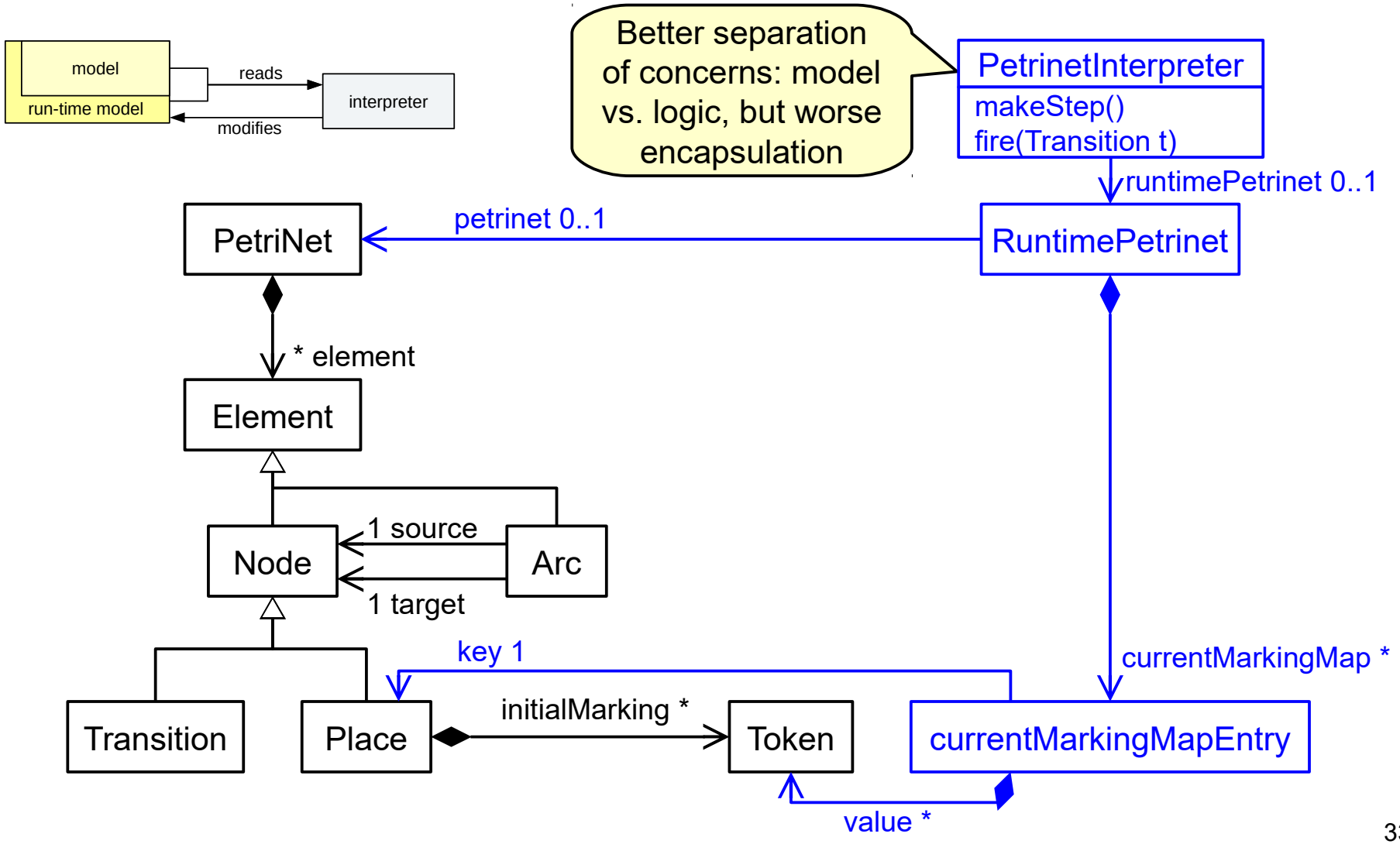
Interpreter and Run-time Model Extension

- Example: Petri net interpreter part of the runtime model



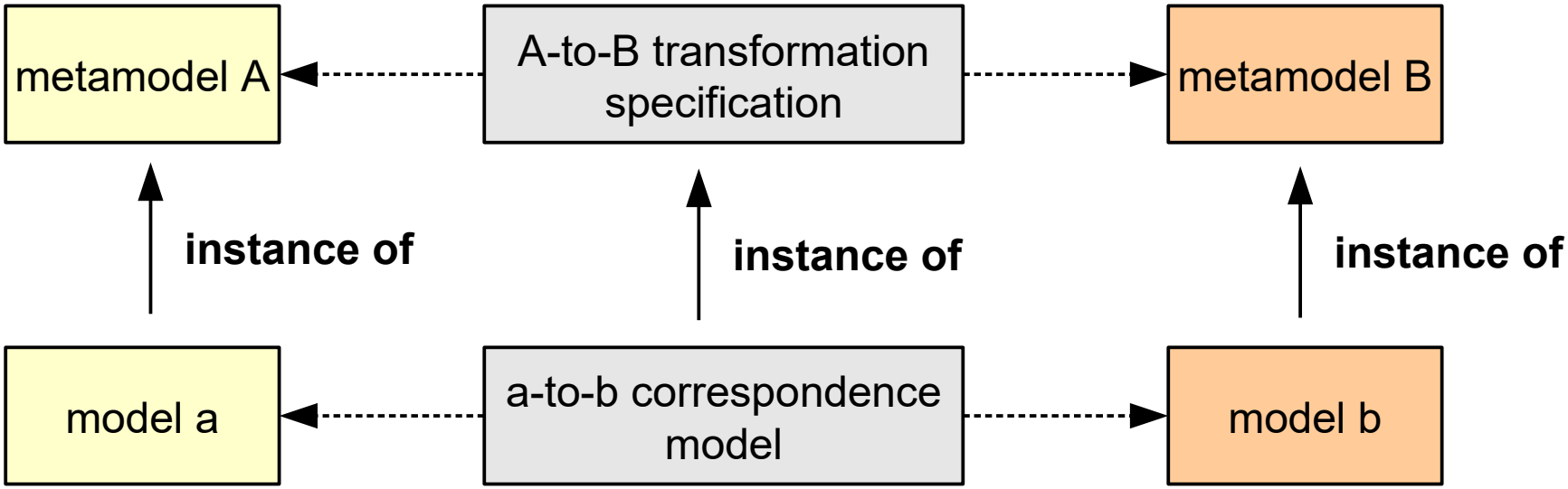
Interpreter and Run-time Model Extension

- Example: Petri net interpreter externally



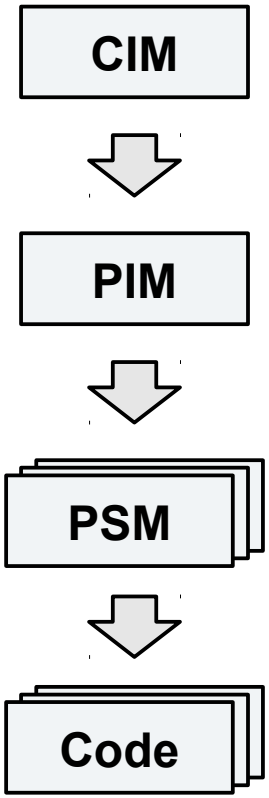
Model-to-Model Transformations

- A typical way to view model-to-model transformations
  - transformation from language A to language B
  - the transformation specification refers to metamodels A and B
  - sometimes: the transformation creates a *correspondence model* of how specifically elements of model a and b relate



Model-Driven Architecture (MDA)

- OMGs **Model-Driven Architecture (MDA)** defines a framework for model-driven software development
- MDA suggest to have the following kinds of models and transformations
  - Computation Independent Model (CIM)
    - "domain model", free of implementation/ technology details
  - Platform Independent Model (PIM)
    - "focuses on the operational capabilities of a system outside the context of a specific platform" (Fast Guide to MDA)
  - Platform Specific Model (PSM)
    - contains details on how to operate on a specific platform



SOFTWARE ENGINEERING

5.2. Model-to-text transformation (code generation)

Leibniz Universität Hannover

Model-to-Text Transformations

- Model-2-Text transformations transform models into code or code-like artifacts
  - Java code
  - XML-Files
  - Configuration files
  - Documentation...

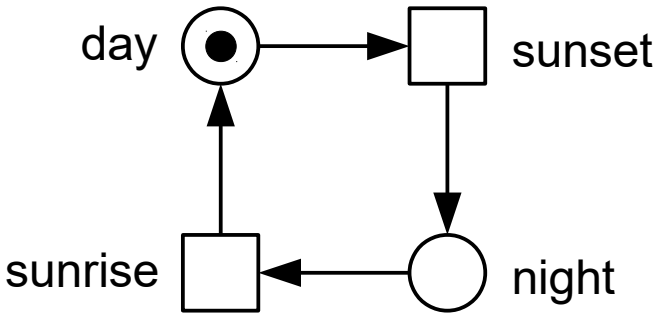
- This is a special case of model-2-model transformation that deserves special treatment and techniques.

Model-to-Text Transformations

- Different ways to realize Model-to-Text Transformations
  - Program transformation using Java / Xtend
  - use a template engine
    - Jet, Velocity, XPand, …

Example: Petrinet to Java

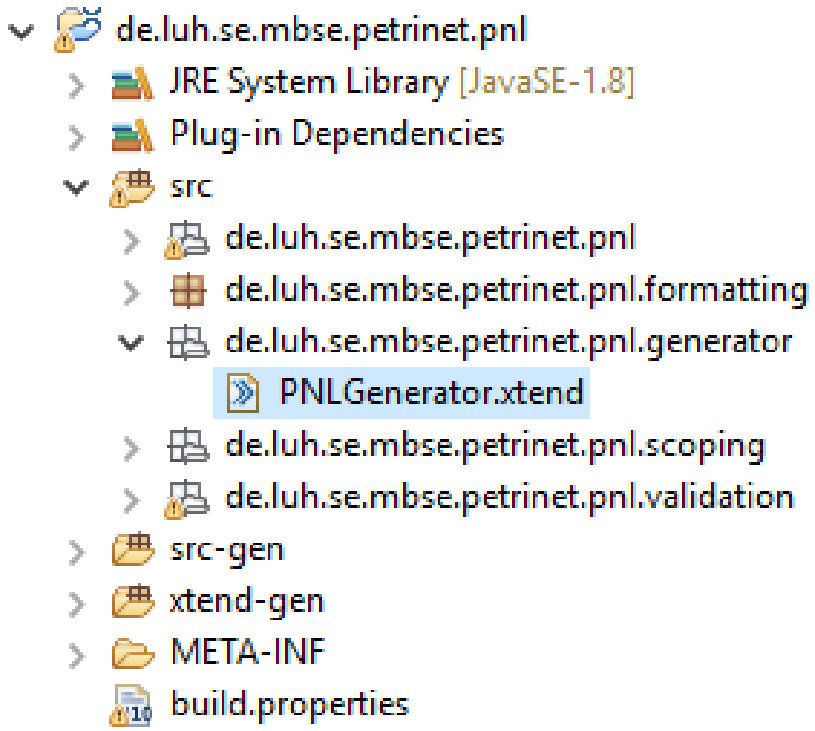
- Example:




```
public class DayAndNight {
      // places
      int day=1; int night=0;
      // main makeStep method
      public void makeStep(){
            if (canFireSunset()){
                  doFireSunset()
            } else
            if (canFireSunrise()){
                  doFireSunrise()
            } else
            { System.out.println("Cannot fire");}
      }
      // transition's canFire and doFire methods
      protected boolean canFireSunset(){
            return (day > 0);
      }
      protected void doFireSunset(){
            day--; night++;
      }
      protected boolean canFireSunrise(){
            return (night > 0);
      }
      protected void doFireSunrise(){
            night--; day++;
      }
}
```

Xtext and XPand

- Every Xtext project has an empty code generator



Xtext and XPand

- Every Xtext project has an empty code generator

```
10
11 /**
12  * Generates code from your model files on save.
13  *
14  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
15  */
16 class PNLGenerator extends AbstractGenerator {
17
18     override void doGenerate(Resource resource,
19         IFileSystemAccess2 fsa,
20         IGeneratorContext context
21     ) {
22 //      fsa.generateFile('greetings.txt', 'People to greet: ' +
23 //          resource.allContents
24 //              .filter(typeof(Greeting))
25 //              .map[name]
26 //              .join(', '))
27     }
28 }
29
```

Xtext and XPand

- We can implement our custom code generator for example as follows:

```
22    override void doGenerate(Resource resource,
23        IFileSystemAccess2 fsa,
24        IGeneratorContext context
25    ) {
26        for (pn : resource.allContents.toIterable.filter(Petrinet)){
27            fsa.generateFile(
28                "petrinets/" + pn.name + ".java",
29                pn.compile
30            )
31        }
32    }
33
34    def compile(Petrinet pn){
35        '''
36        package petrinets;
37
38        public class «pn.name» {
39
40            // places
41            «FOR p : pn.element.filter(Place)»
42                «p.compile»
43            «ENDFOR»
44            // main makeStep method
45            public void makeStep(){
46                «FOR t : pn.element.filter(Transition)»
47                    «t.compileForMakeStep»
48                «ENDFOR»
49                { System.out.println("Cannot fire");}
50            }
51
52            // transition's canFire and doFire methods
53            «FOR t : pn.element.filter(Transition)»
54                «t.compile»
55            «ENDFOR»
56        }
57        '''
58    }
```

...