

# Model-Based Software Engineering

Lecture 05 – Concrete Syntax

*Prof. Dr. Joel Greenyer*



May 3, 2016



# Acknowledgment

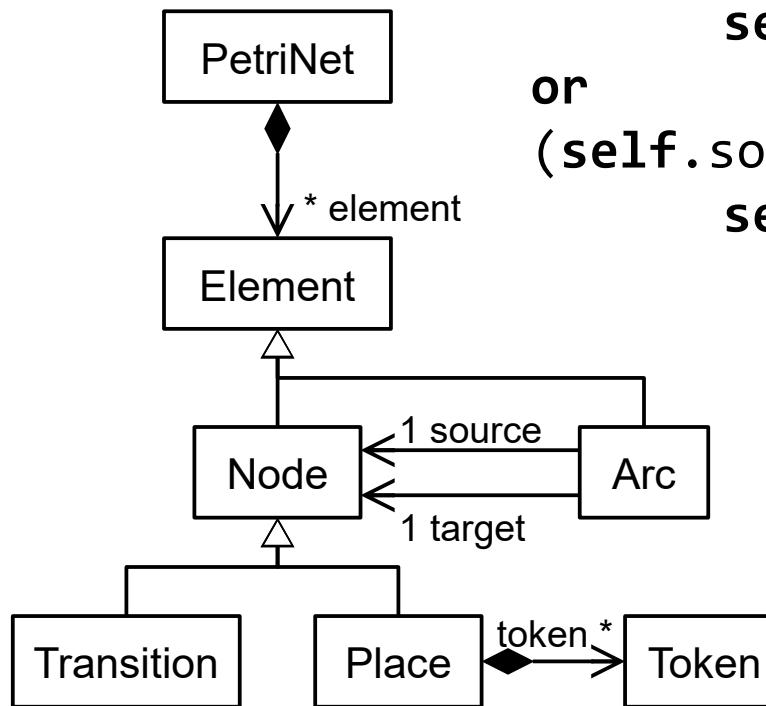
- The slides of this lecture are inspired by lecture slides from
  - *Ekkart Kindler*: Course on Advanced Topics in Software Engineering, DTU Compute, 2015.
    - <http://www2.imm.dtu.dk/courses/02265/f15/schedule.shtml>
  - *Ina Schäfer, Christoph Seidl*: Modellbasierte Softwareentwicklung, TU Braunschweig,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

# OCL – Example

in the last lecture...

- Invariant constraint written in the **Object Constraint Language (OCL)**:

```
context Arc
inv "No Arcs Between Nodes Of The Same Kind":
((self.sourceoclIsKindOf(Place) and
  self.targetoclIsKindOf(Transition))
or
(self.sourceoclIsKindOf(Transition) and
  self.targetoclIsKindOf(Place) ) );
```



# OCLinEcore Editor

## Example: Company

in the last lecture...

```
company.ecore X
1 import ecore : 'http://www.eclipse.org/emf/2002/Ecore' ;
2
3 package company : company = 'http://www.example.org/company'
4 {
5     class Company extends NamedElement
6     {
7         property department : Department[*] { ordered composes };
8     }
9     class Department extends NamedElement
10    {
11        property employee : Person[*] { ordered composes };
12        attribute ageSumOfEmployees : ecore::EInt[?] { derived readonly transient volatile }
13        {
14            initial: self.employee->iterate(p; sum:Integer = 0 | sum + p.age);
15        }
16    }
17    class NamedElement
18    {
19        attribute name : String[?];
20    }
21    class Person extends NamedElement
22    {
23        attribute age : ecore::EInt[?];
24        invariant AllEmployeesMustBeAdults: self.age >= 18;
25    }
26 }
```

derived attribute

invariant

# OCLinEcore Editor

## Example: Company

in the last lecture...

The screenshot shows the OCLinEcore Editor interface. On the left is the Model Explorer view, which displays the project structure. It includes a 'Model Dependencies' section with 'Project Dependencies' (src, JRE System Library [JavaSE-1], Plug-in Dependencies, META-INF), a 'model' section containing 'company.aidr', 'company.ecore', 'company.ecore.oclas', 'company.genmodel', and 'Company.xmi', and another 'Model Dependencies' section for 'de.luh.se.mbsse.petrinet'. The 'Outline' view below it shows 'outline is not available.' In the center, there are two tabs: 'company.ecore' and 'Company.xmi'. The 'Company.xmi' tab is active, showing a tree structure of a company model. It includes a 'Company ModelingInc' node with 'Department Finance' (Person Alison, Person Beverly) and 'Department Marketing' (Person Charlie, Person Dave). A red error icon is next to the 'Company ModelingInc' node. Below the tree is a 'Properties' view showing two properties: 'Age' with value '17' and 'Name' with value 'Dave'. A yellow callout bubble points from the 'Name' row to the right, containing the text 'validation shows invalid value'. At the bottom right of the Properties view, there are several icons for navigating between tabs and views.

Property	Value
Age	17
Name	Dave

# OCLinEcore Editor

## Example: Company

in the last lecture...

The screenshot shows the OCLinEcore Editor interface. On the left, the Model Explorer view displays a project structure for 'de.luh.se.mbsse.company' containing 'src', 'JRE System Library [JavaSE-1]', 'Plug-in Dependencies', 'META-INF', and a 'model' folder with files like 'company.ajrd', 'company.ecore', 'company.ecore.oclas', 'company.genmodel', and 'Company.xmi'. The 'Outline' view below it is empty. The central workspace shows a tree view of the 'company.ecore' model, which includes a 'Company ModelingInc' package with 'Department Finance' and 'Department Marketing' sub-packages. 'Department Finance' contains 'Person Alison' and 'Person Beverly'. 'Department Marketing' contains 'Person Charlie' and 'Person Dave'. A separate 'platform:/resource/de.luh.se.mbsse.company/model/company.ecore' entry is also visible. At the bottom, the Properties view shows two properties: 'Age Sum Of Employees' with a value of 61, and 'Name' with a value of 'Marketing'. A yellow callout bubble points from the text 'interpretation of OCL derived value specifications on dynamic instance model' to the 'Properties' view.

Quick Access | Java | Model

Model Explorer X

type filter text

de.luh.se.mbsse.company

- Project Dependencies
- src
- JRE System Library [JavaSE-1]
- Plug-in Dependencies
- META-INF
- model
  - company.ajrd
  - company.ecore
  - company.ecore.oclas
  - company.genmodel
  - Company.xmi
- de.luh.se.mbsse.petrinet

Outline X

in outline is not available.

company.ecore Company.xmi

platform:/resource/de.luh.se.mbsse.company/model/Company.xmi

- Company ModelingInc
  - Department Finance
    - Person Alison
    - Person Beverly
  - Department Marketing
    - Person Charlie
    - Person Dave

platform:/resource/de.luh.se.mbsse.company/model/company.ecore

Properties X Problems

Property	Value
Age Sum Of Employees	61
Name	Marketing

interpretation of OCL derived value specifications on dynamic instance model

# Model-Based Software Engineering

## Lecture 05 – Concrete Syntax

*Prof. Dr. Joel Greenyer*



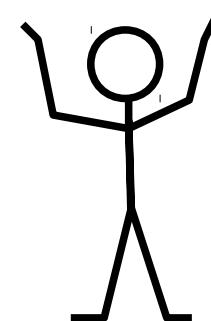
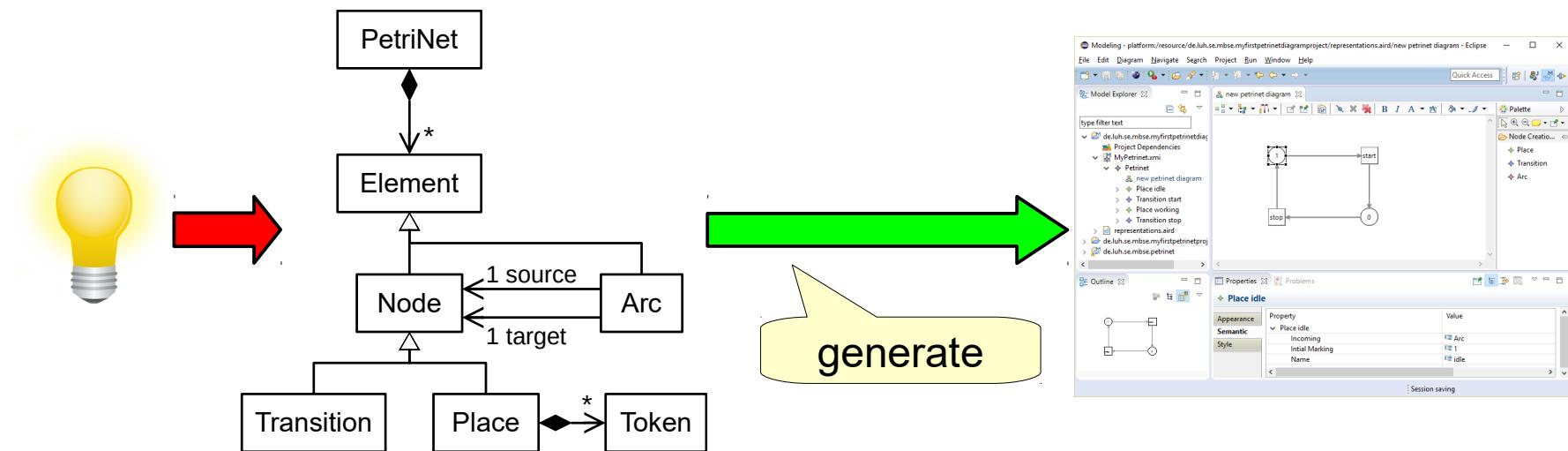
May 3, 2016



# Build a Petri Net Modeling Tool

in a previous lecture...

- Model-based approach for building modeling tools: Provide only a few conceptual models and generate tool automatically



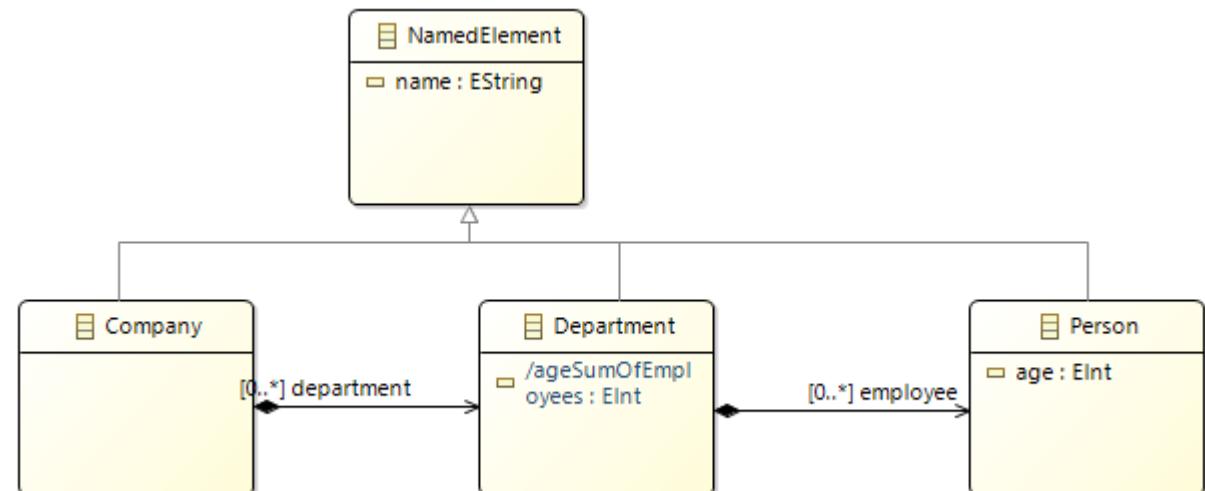
# Concrete Syntax: Textual or Graphical

- Example: textual vs. graphical syntax for Ecore

```
package company : company = 'http://www.example.org/company' {
    class Company extends NamedElement{
        property department : Department[*] { ordered composes } ;
    }
    class Department extends NamedElement{
        property employee : Person[*] { ordered composes } ;
    }
    class NamedElement{
        attribute name : String[?];
    }
    class Person extends NamedElement{
        attribute age : ecore::EInt[?];
    }
}
```

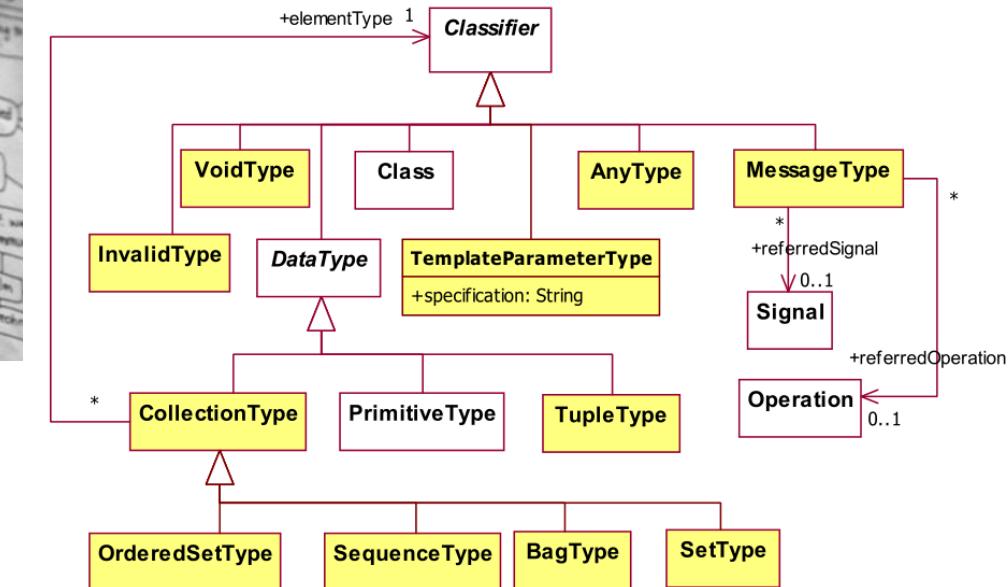
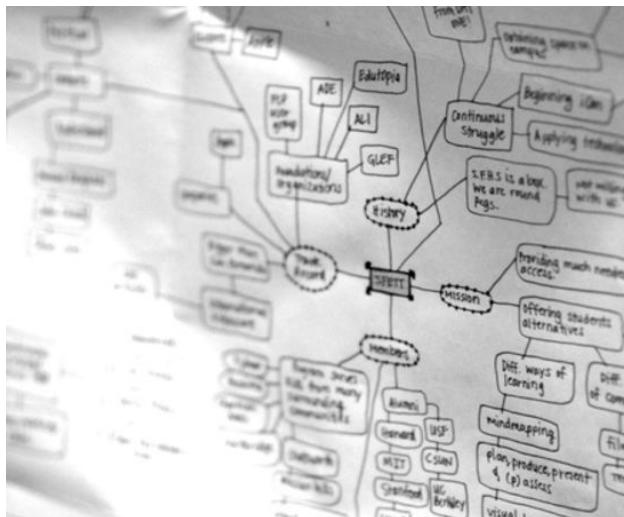
*textual Ecore  
class model  
syntax*

*graphical Ecore  
class model  
("class diagram")  
syntax*



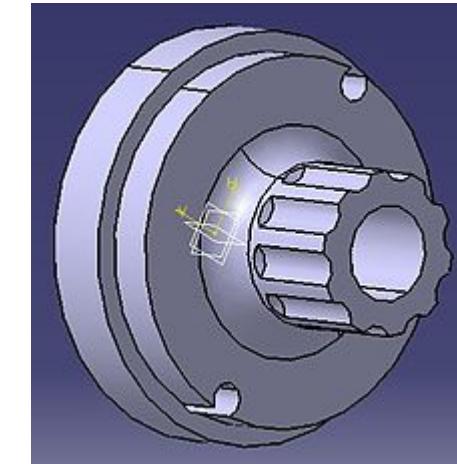
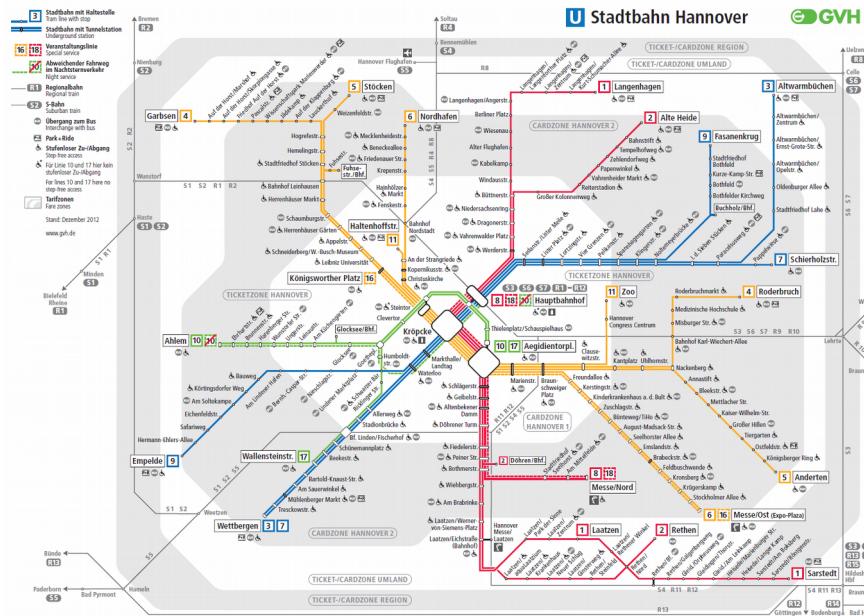
# Textual vs. Graphical Syntax

- **Benefits of graphical syntax (vs. textual syntax):**
  - **Objects-and-relationships concepts (graphs)** are easier understand when displayed graphically
    - mind maps, organigrams, class diagrams, etc.



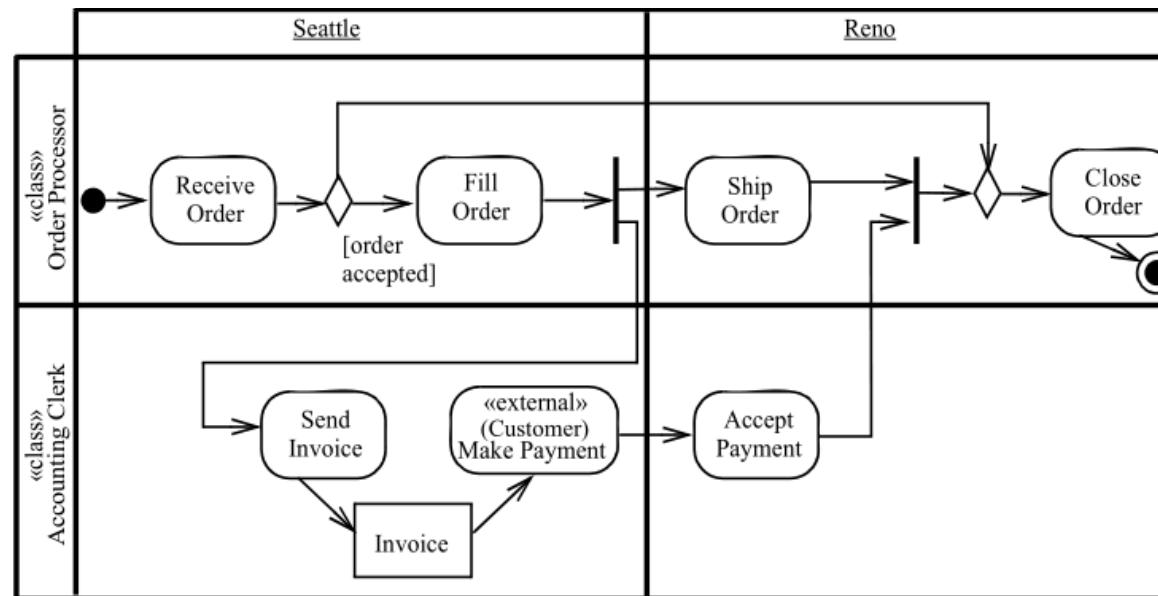
# Textual vs. Graphical Syntax

- **Advantages of graphical syntax (vs. textual syntax):**
  - Models that capture **spatial aspects** are best understood graphically
    - (street) maps, floor plans, fire escape route plans
    - even 3D syntax, for example for CAD



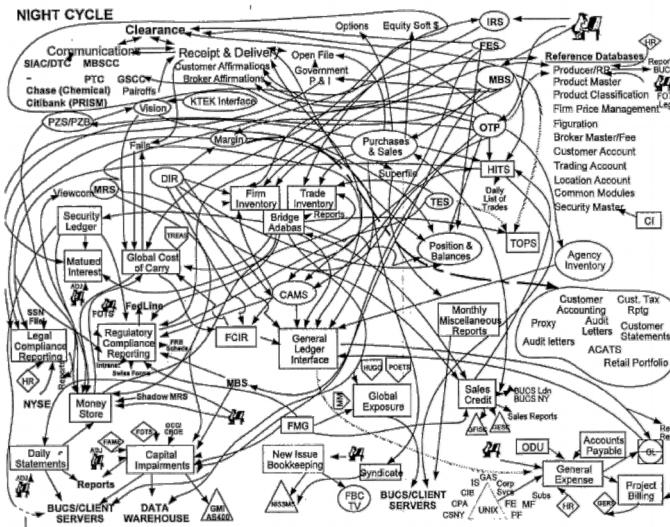
# Textual vs. Graphical Syntax

- **Advantages of graphical syntax (vs. textual syntax):**
  - use of **colors or shapes** to highlight different concepts
  - use of **two dimensions** to express different aspects
    - example: UML Activity Diagram with multidimensional swimlanes



# Textual vs. Graphical Syntax

- **Advantages of graphical syntax (vs. textual syntax):**
  - may become **confusing** for displaying large models
  - it may become **hard to maintain a nice layout**
    - there exist automatic layout algorithms, but often they do not yield the desired result



# Textual vs. Graphical Syntax

---

- **Advantages of graphical syntax (vs. textual syntax):**
  - **graphical editors** usually take more effort to build
  - especially, good **usability** for **graphical editors** is more challenging to achieve
    - for example: editing graphical elements with text often requires switching between mouse and keyboard
  - also, **graphical editors** may not have good **performance**
    - when editing, loading/saving
  - versioning (diff/merge) is more difficult for graphical models
- Do you see other advantages/disadvantages?

# Textual vs. Graphical Syntax

- **Advantages of textual syntax** (vs. graphical syntax):
  - easy cut/copy/paste
  - easy versioning (diff/merge)
  - clear information flow (reading direction)
  - fast editing “straight on typing”
- **Disadvantages of textual syntax** (vs. graphical syntax):
  - complex dependencies are not easy to see  
(because of indirect references)

```
class Department extends NamedElement{
    property employee : Person[*] { ordered composes };
}
...
class Person extends NamedElement{
    attribute age : ecore::EInt[?];
}
```



## 4.1. *Graphical syntax*

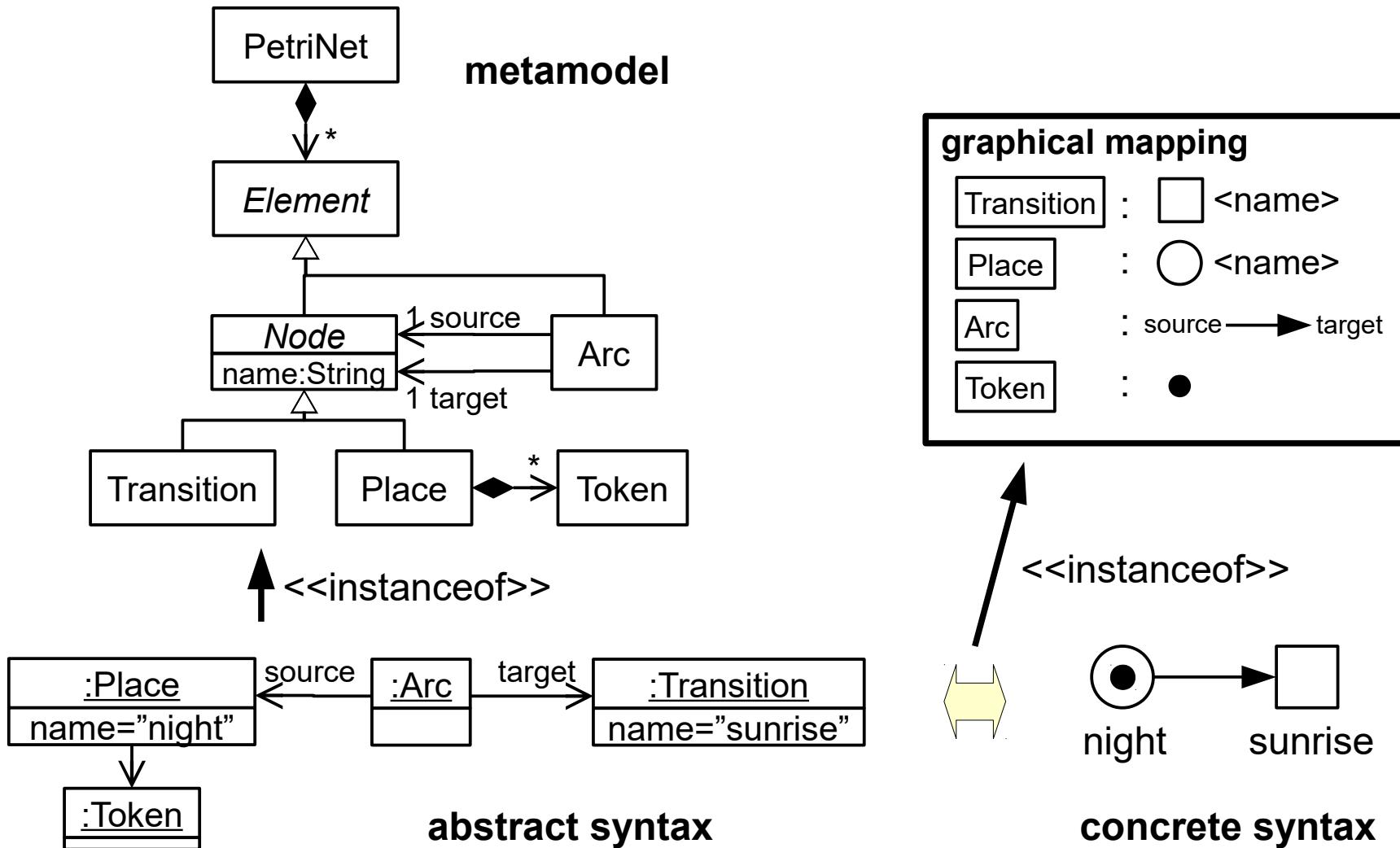
# Building a Graphical Language – General Steps

---

- 1) Specify a metamodel (that defines the language's abstract syntax)
- 2) Define how abstract syntax elements should be represented visually
- 3) Implement or generate the graphical editor

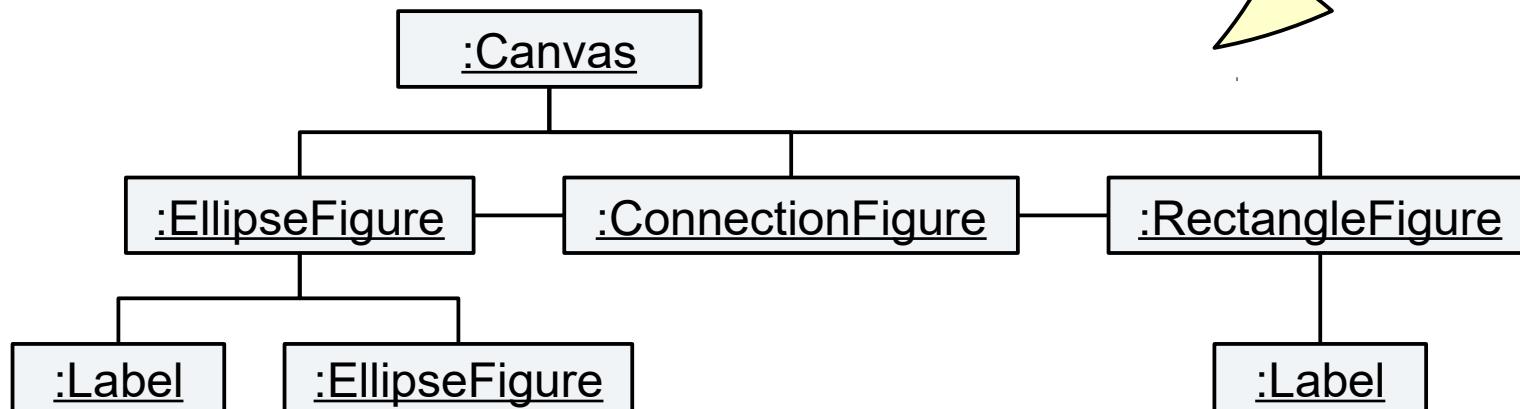
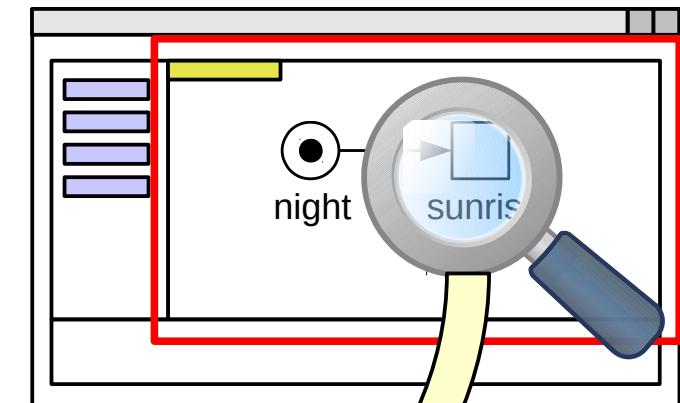
# Build a Graphical Petri Net Language

- Metamodel and mapping of abstract to concrete syntax



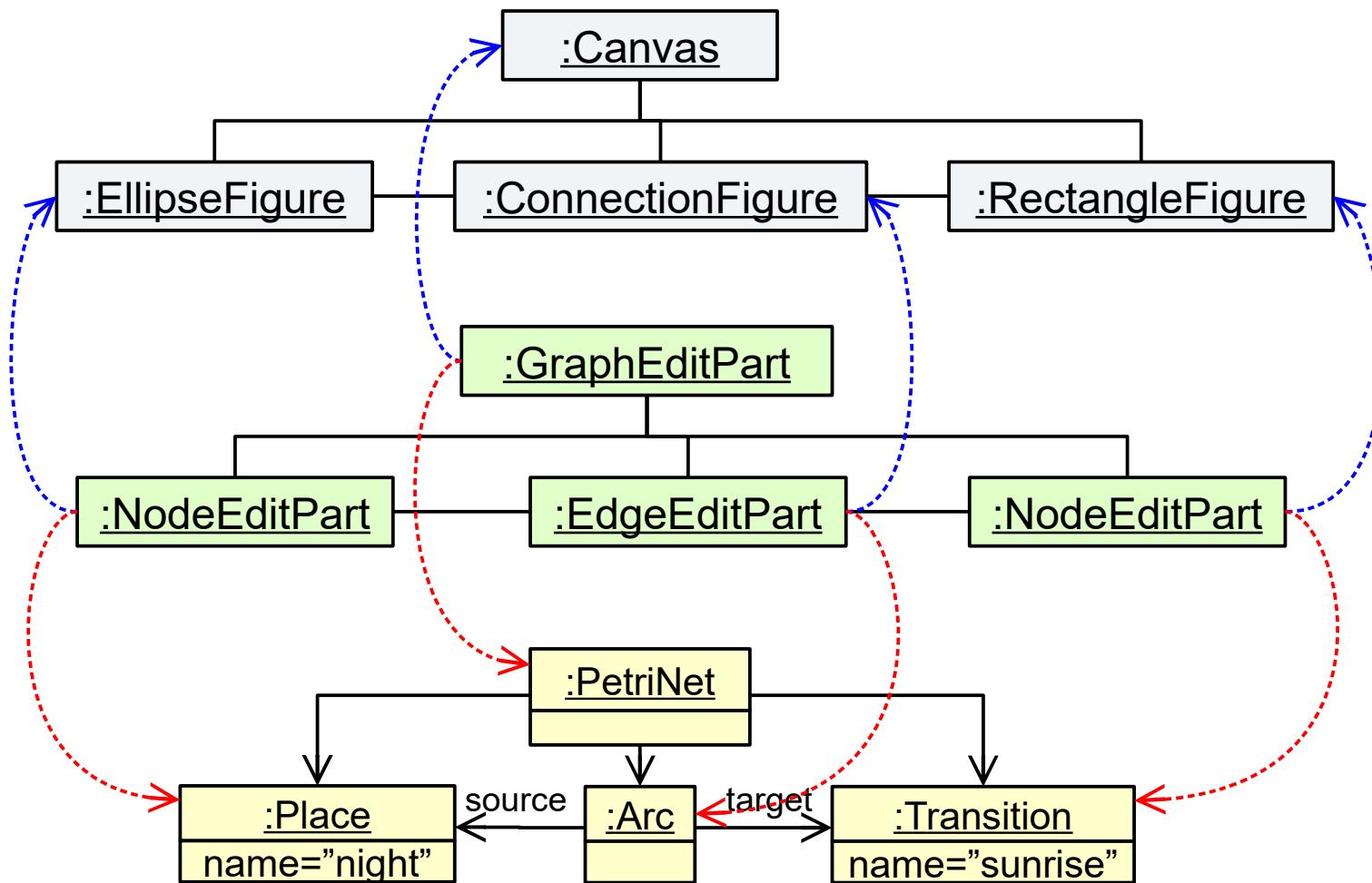
# Implementing a Graphical Editor Manually

- Implementing the graphics
  - for example using Eclipse GEF (Graphical Editing Framework)



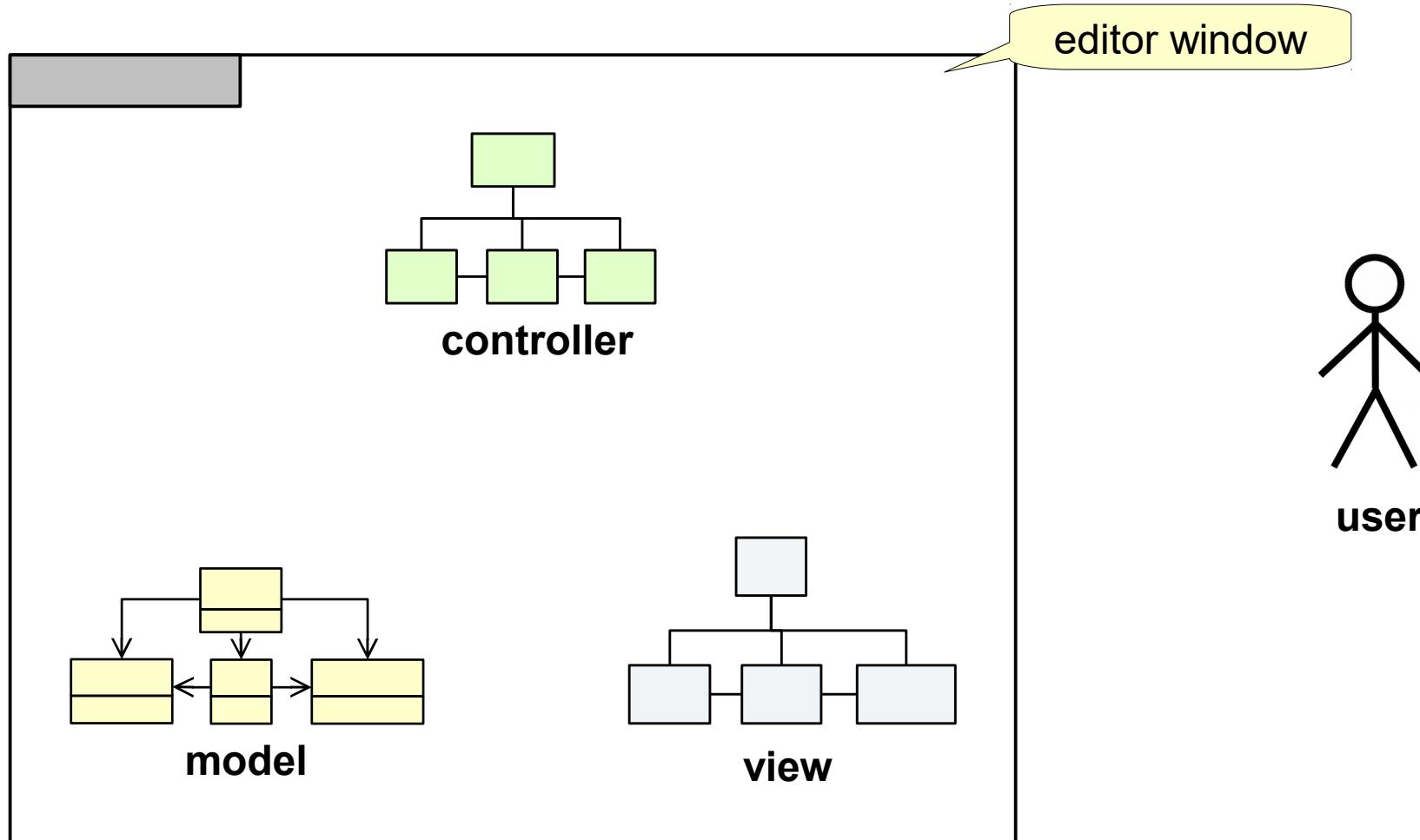
# Implementing a Graphical Editor Manually

- How are graphical elements and model elements connected?
  - typically using the **Model-View-Controller (MVC)** pattern



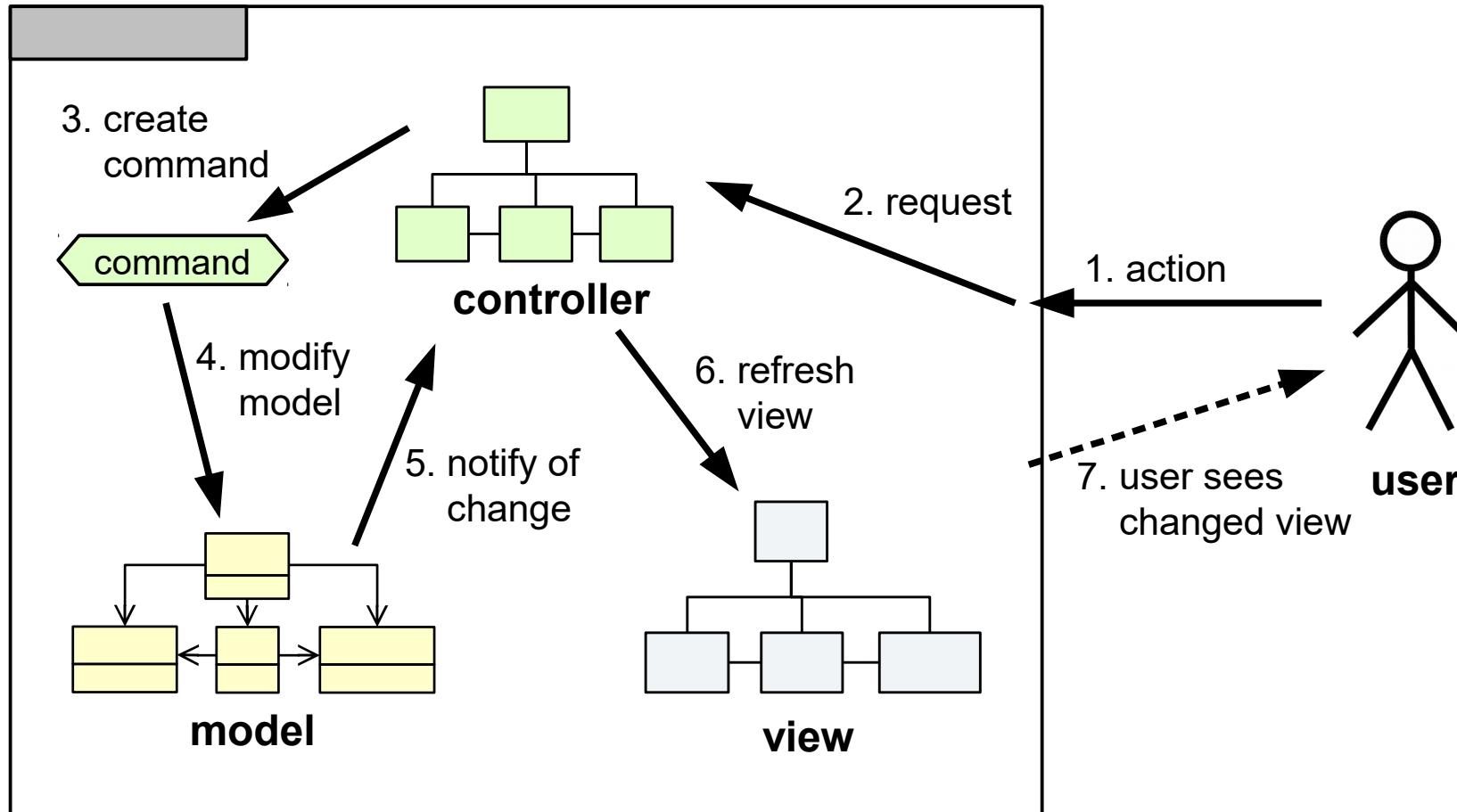
# Implementing a Graphical Editor Manually

- MVC pattern as implemented with GEF



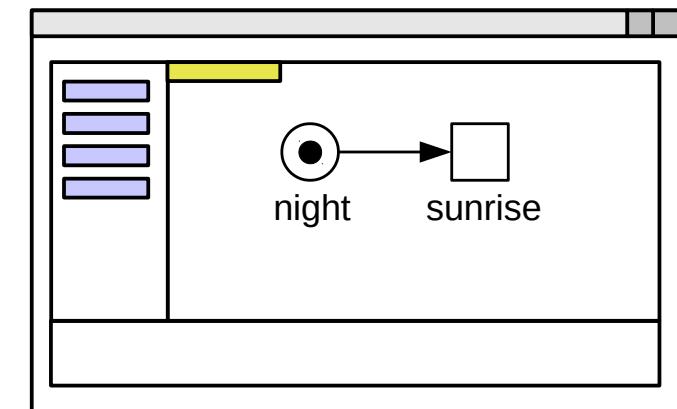
# Implementing a Graphical Editor Manually

- MVC pattern as implemented with GEF



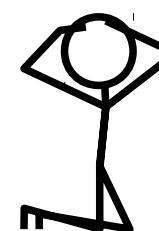
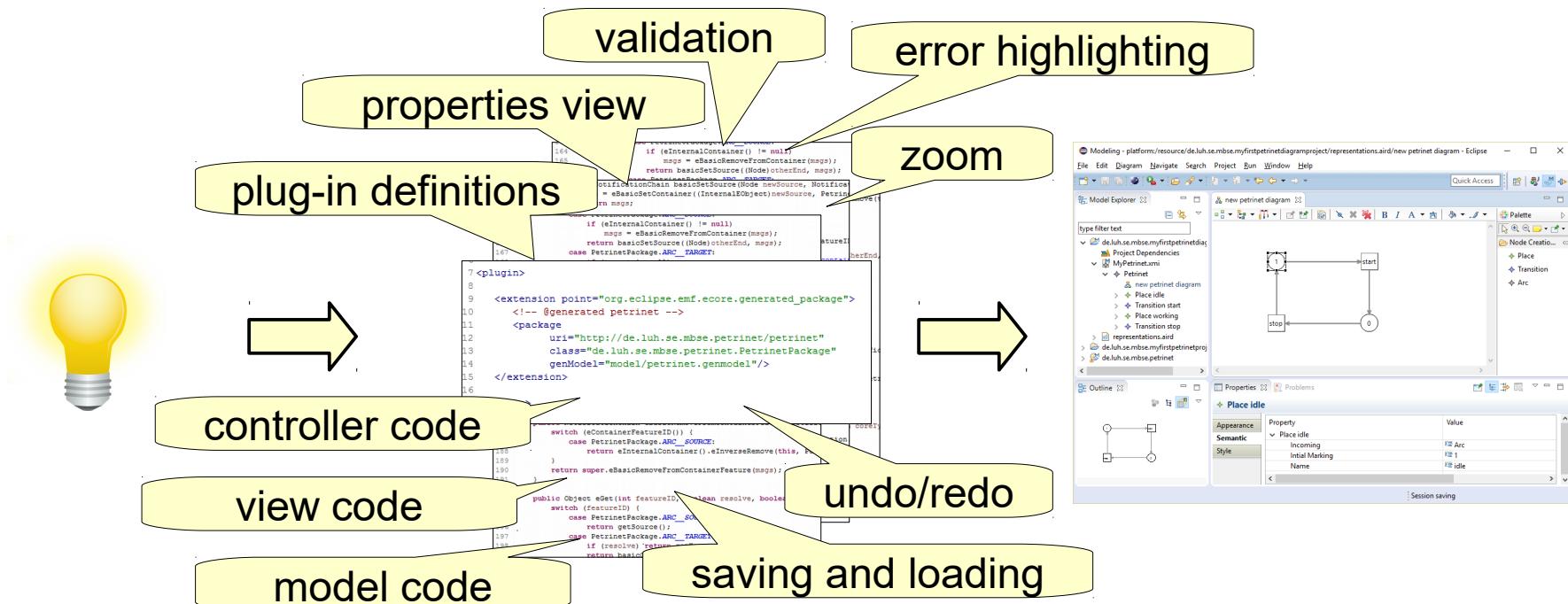
# Implementing a Graphical Editor Manually

- What else must be implemented?
  - undo/redo logic
  - tool bar
  - zoom
  - grid
  - properties view
  - error highlighting
  - ...



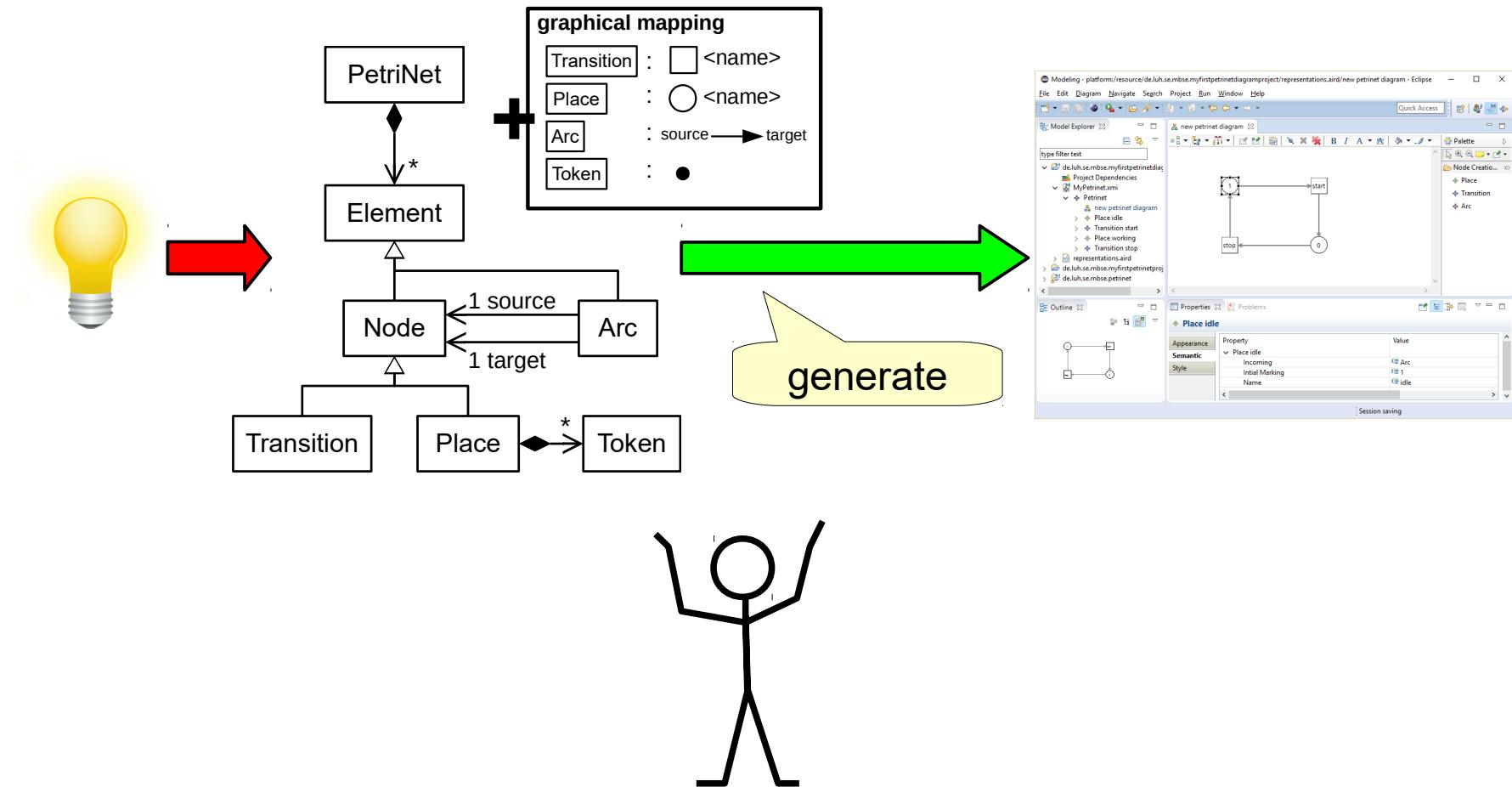
# Generating a Graphical Editor Automatically

- Instead of doing all the work manually...



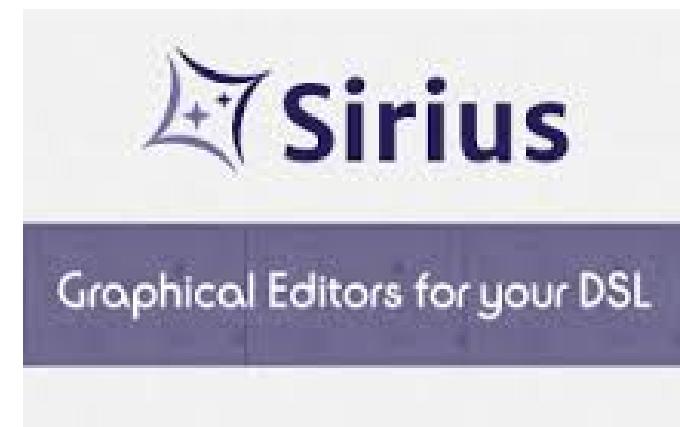
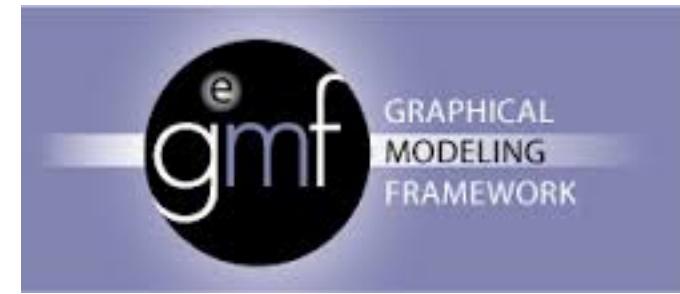
# Generating a Graphical Editor Automatically

- ... we would like to do as much as possible automatically



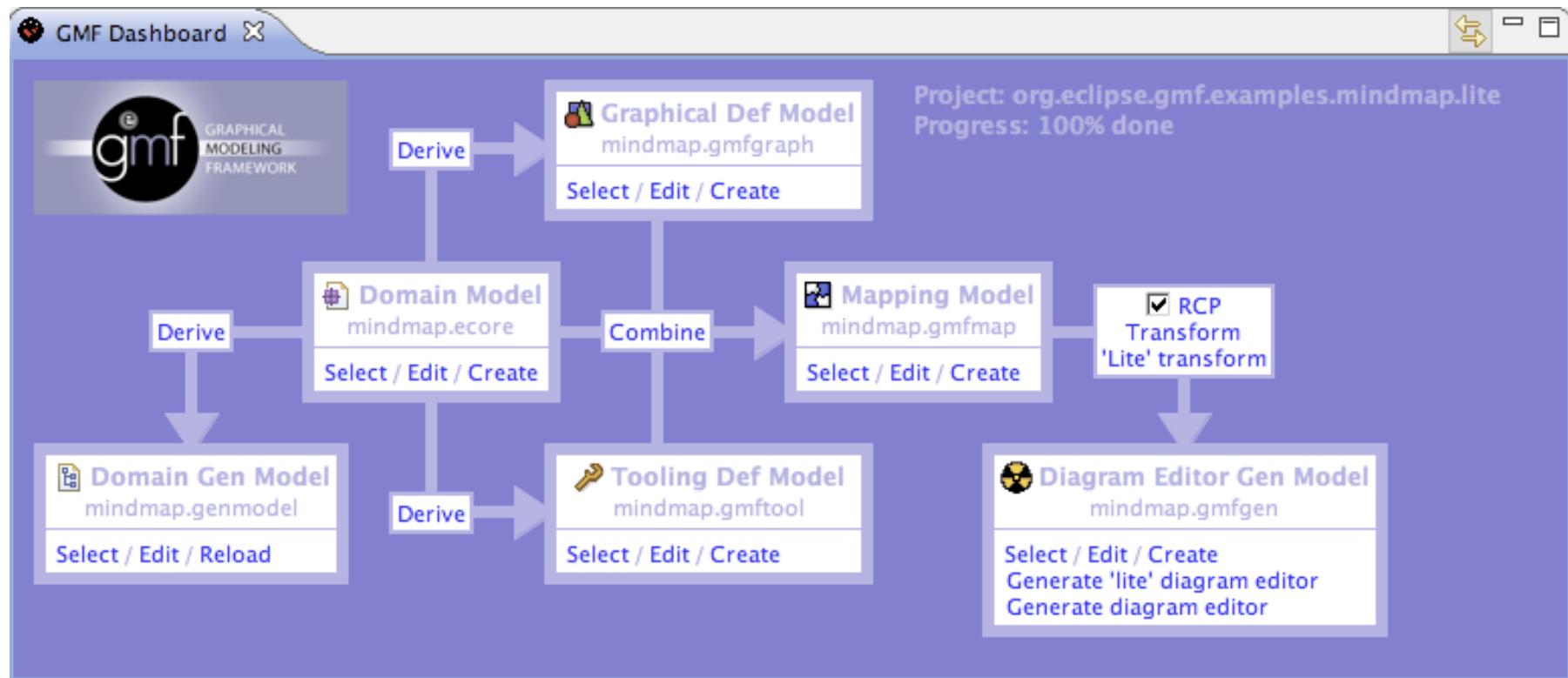
# Eclipse Frameworks for the Automatic Generation/Creation of Graphical Editors

- Graphical Modeling Framework (GMF)
- Graphiti
- Sirius



# Graphical Modeling Framework (GMF)

- Model graphical elements, graphical mapping, tools, ...
- Then generate editor code from the combined models

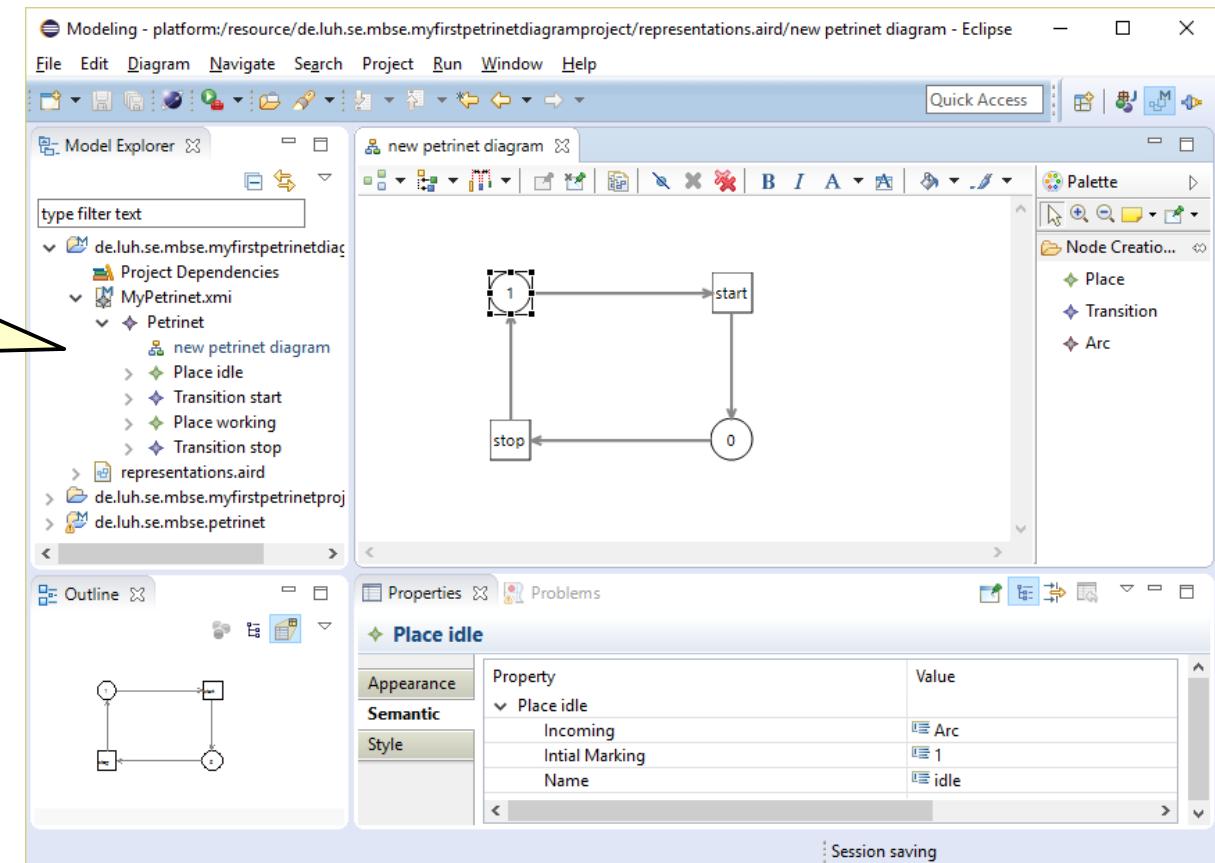


see <http://www.eclipse.org/modeling/gmp/>  
and [https://wiki.eclipse.org/Graphical\\_Modeling\\_Framework/Tutorial/Part\\_1](https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1)

# Eclipse Sirius

- Eclipse **Sirius** works by *interpreting* a graphical mapping of the model
  - no code generation required

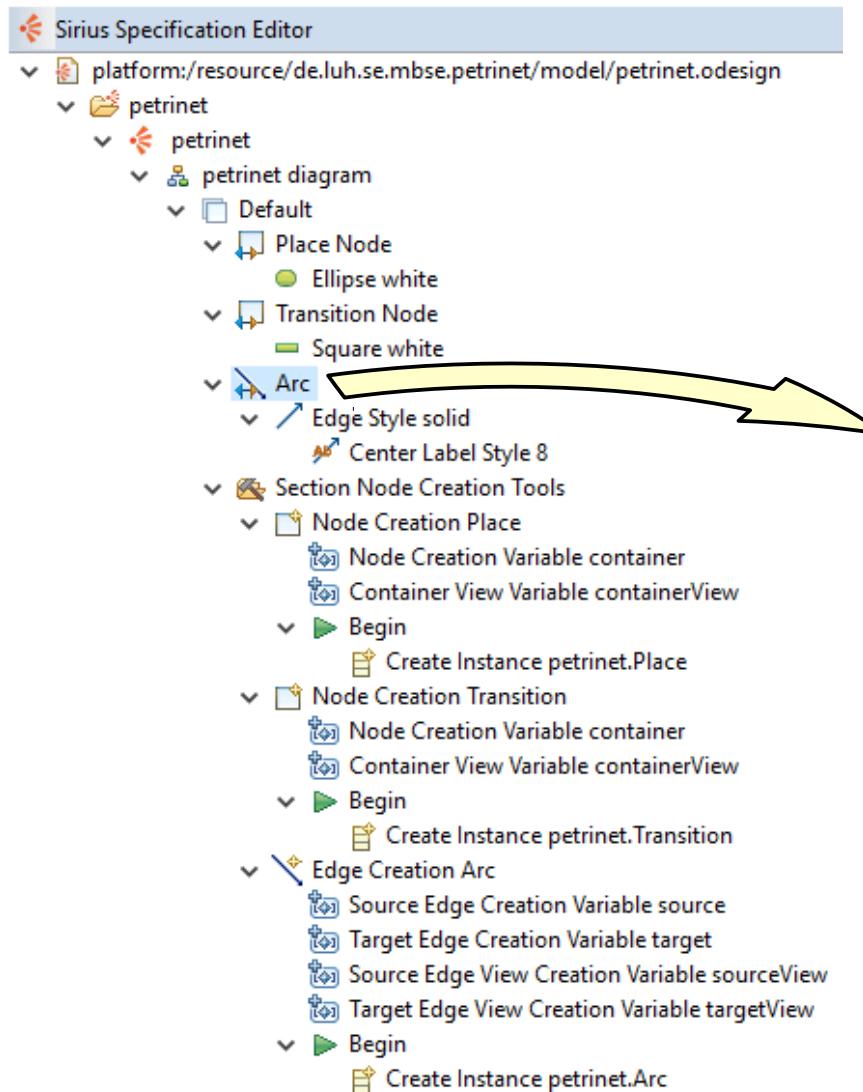
**example:** Petri net editor created in 15 Minutes!



see <https://eclipse.org/sirius/>  
and <https://eclipse.org/sirius/getstarted.html>

# Eclipse Sirius

- Sirius graphical mapping:



defining how an arc connector should connect source and target elements (via its features “source” and “target”)

**Id\*:**

Arc

**Domain Class\*:**

petrinet.Arc

**Source Mapping\*:**

Place Node, Transition Node

**Source Finder Expression:**

feature:source

**Target Mapping\*:**

Place Node, Transition Node

**Target Finder Expression\*:**

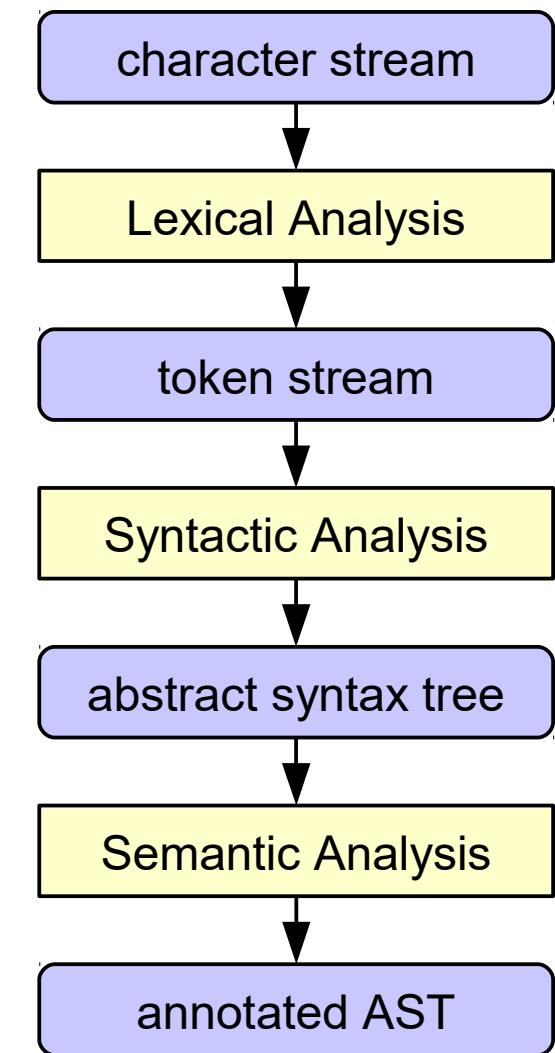
feature:target

**Semantic Candidates Expression:**

## *4.2. Textual syntax*

# Excursion on Compilers – Syntax Analysis

- Steps in the textual syntax analysis:
  - Lexical Analysis
    - partitions character stream into tokens (removes whitespaces, identifies keywords, identifiers, ...)
  - Syntax Analysis
    - context-free analysis identifies abstract syntax tree (AST) structure
  - Semantic Analysis
    - analyze cross-references of AST elements (variable scoping, type conformance, ...)



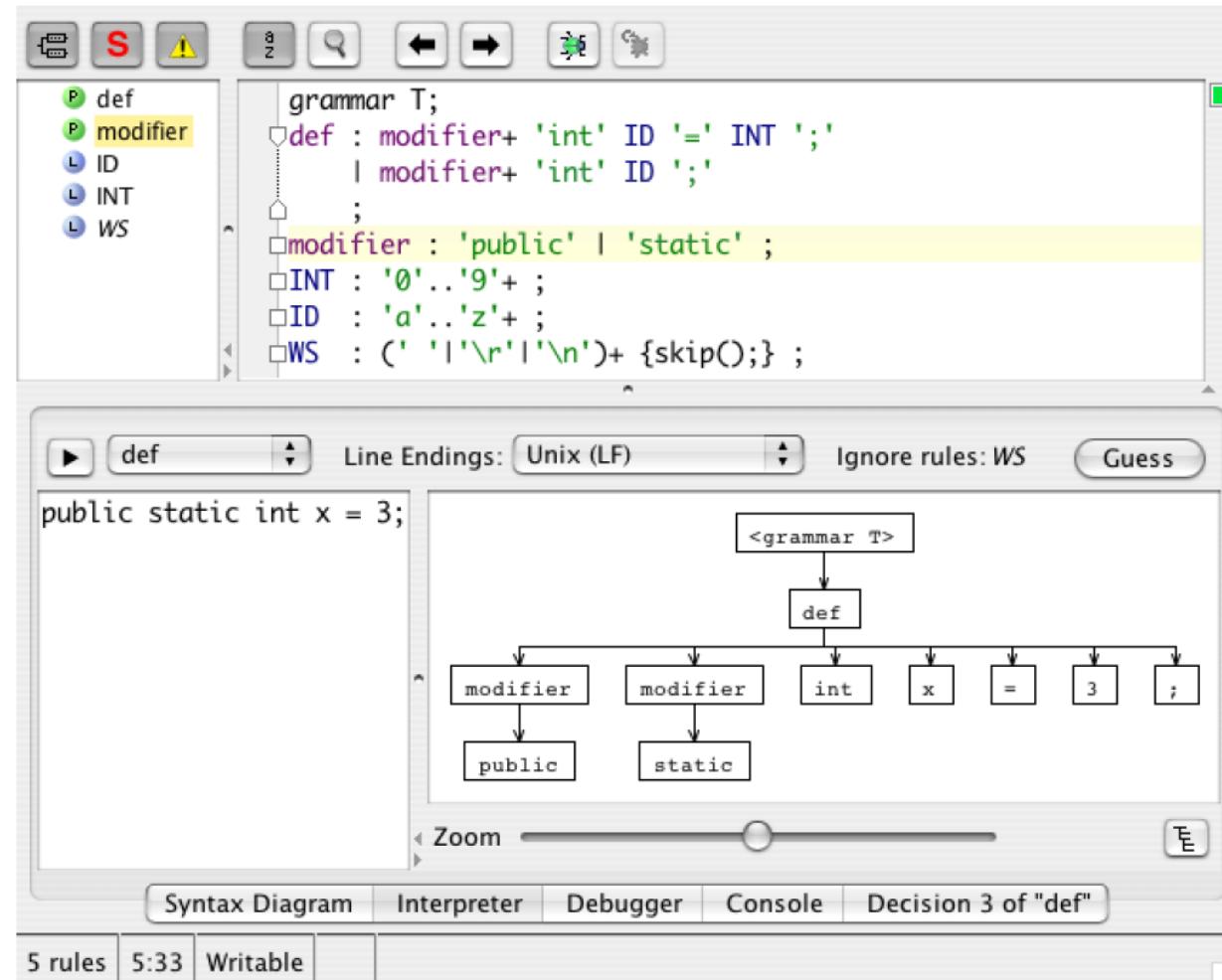
# Parser generators – Example: ANTLR

---

- There exist **frameworks for constructing compilers** that can generate lexer and parser components (and other things) from a language definition in the form of a grammar
- a popular example: **ANTLR**
  - takes as input a context-free grammar in Extended Backus–Naur Form (EBNF)

# Parser generators – Example: ANTLR

- ANTLRWorks Interpreter visualizes the result of the syntactic analysis:



# Xtext – a Framework for Building Textual Languages in Eclipse

---

- **Xtext** is a framework for building textual languages and editors within Eclipse (also IntelliJ IDEA and web browser)
- The language is built based on a grammar definition similar to EBNF
  - but with extra features for referencing a corresponding Ecore metamodel
  - Generation of an Ecore metamodel from a grammar is also supported
- Xtext can create the Ecore metamodel, lexer/parser, and editor from the grammar definition
  - The editor supports syntax checking, highlighting, and code completion, renaming/refactoring, and has extensions for implementing quick-fixes, and other functionality

# Xtext – Simple Example: Company

```
// automatically generated by Xtext
grammar de.luh.se.mbsc.company.cml.CML with org.eclipse.xtext.common.Terminals

import "platform:/resource/de.luh.se.mbsc.company/model/company.ecore"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

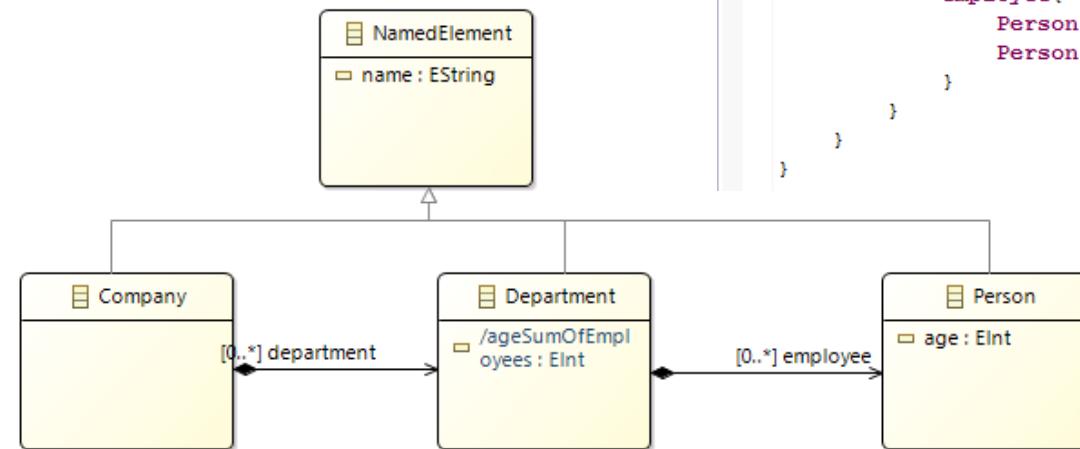
@Company returns Company:
    {Company}
    'Company'
    name=EString
    '{'
        ('department' '{' department+=Department ( "," department+=Department)* '}' )?
    '}';

@EString returns ecore::EString:
    STRING | ID;

@Department returns Department:
    {Department}
    'Department'
    name=EString
    '{'
        ('employee' '{' employee+=Person ( "," employee+=Person)* '}' )?
    '}';

@Person returns Person:
    {Person}
    'Person'
    name=EString
    '{'
        ('age' age=EInt)?
    '}';
}

@EInt returns ecore::EInt:
    '-'? INT;
```



mycompany.cml

```

Company MyCompany{
    department {
        Department Finance{
            employee{
                Person Alison {age 21},
                Person Beverly {age 34}
            }
        },
        Department Marketing{
            employee{
                Person Charlie {age 43},
                Person Dave {age 39}
            }
        }
    }
}
```