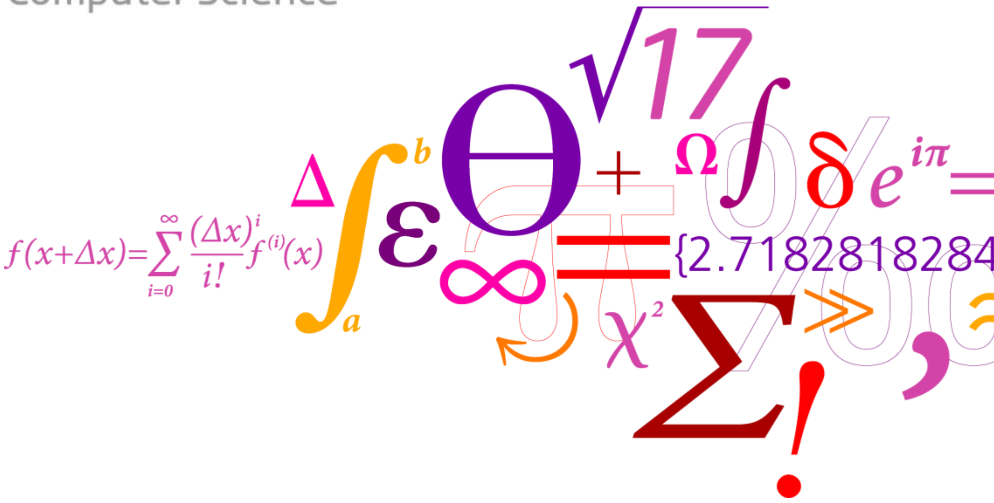
Model-based Software Engineering
(02341, spring 2016)

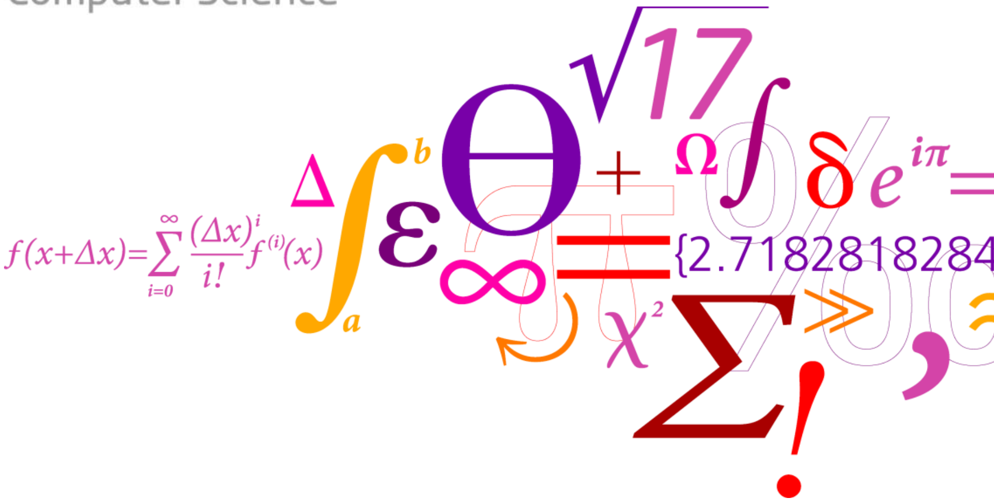
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

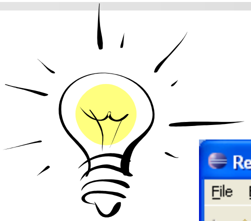


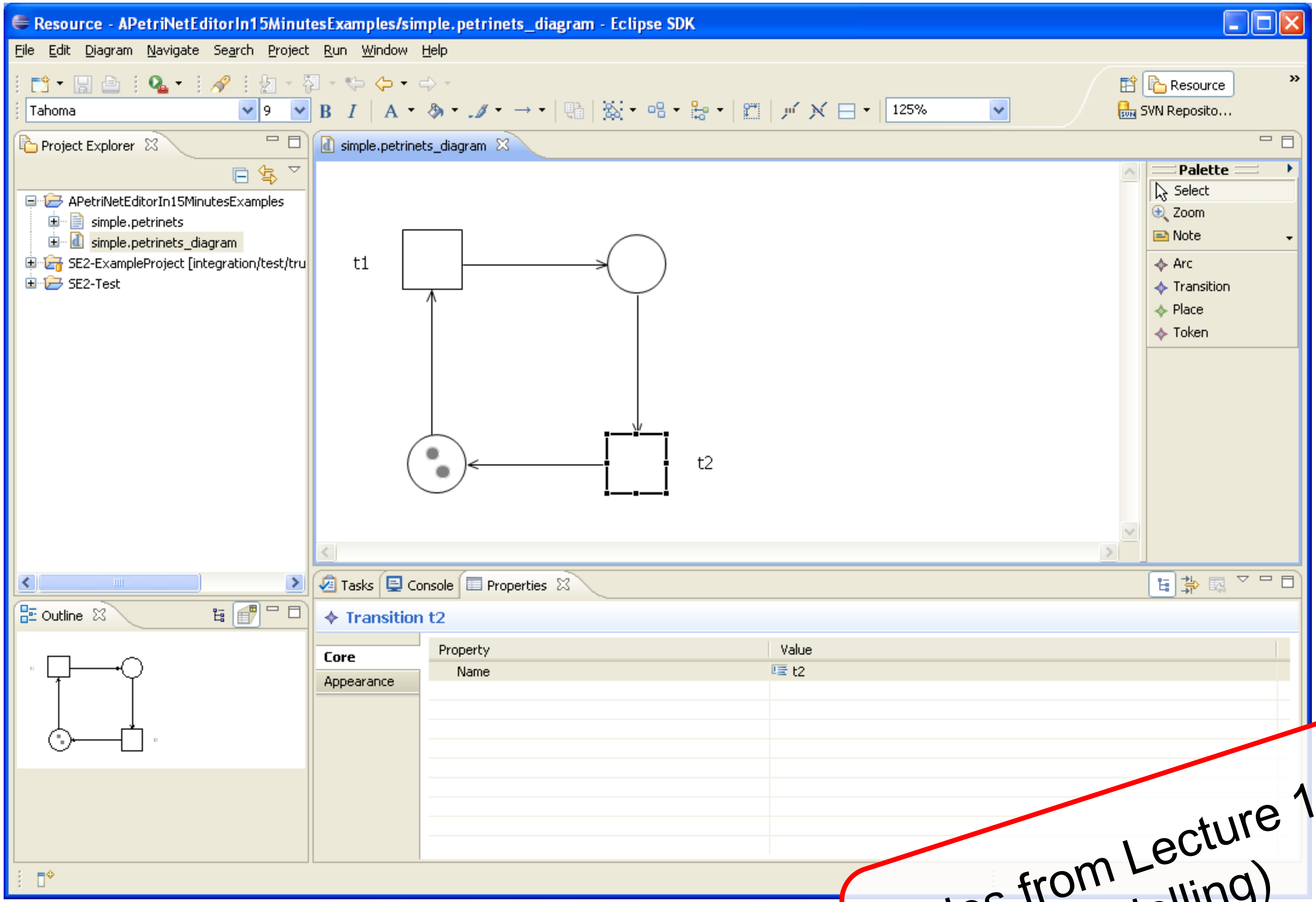
Meta-modelling and
Domain Specific Languages (DSLs)

DTU Compute
Department of Applied Mathematics and Computer Science


$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
{2.7182818284

Idea for some Software

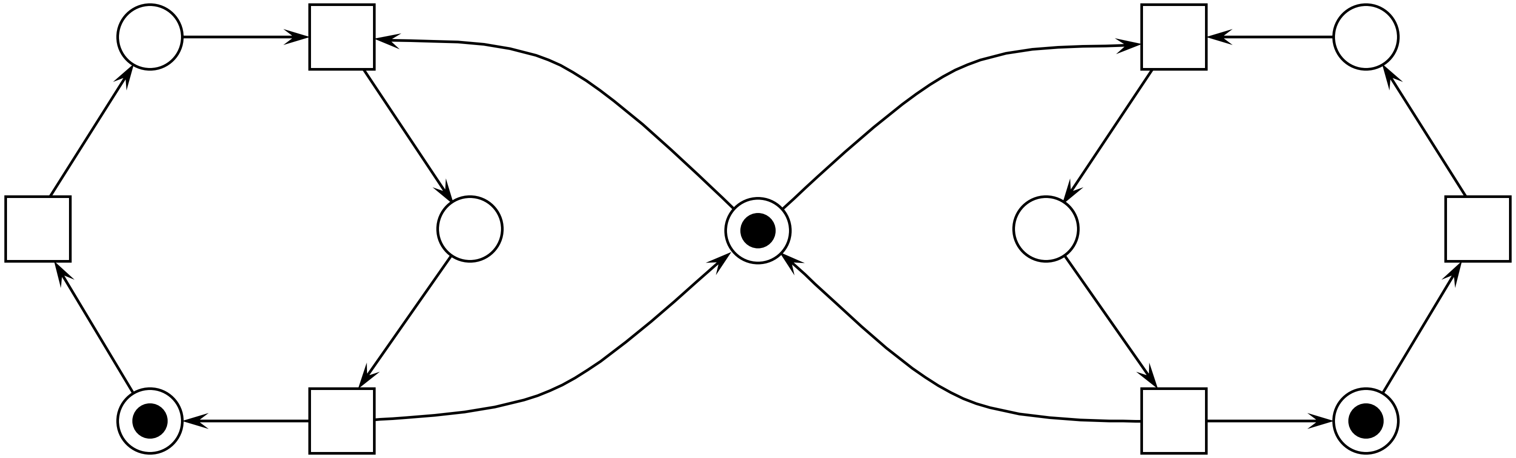




Property	Value
Name	t2

Slides from Lecture 1:
3 – 10 (modelling)

Example of a Petri net

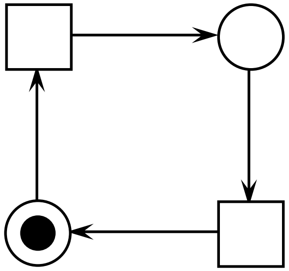


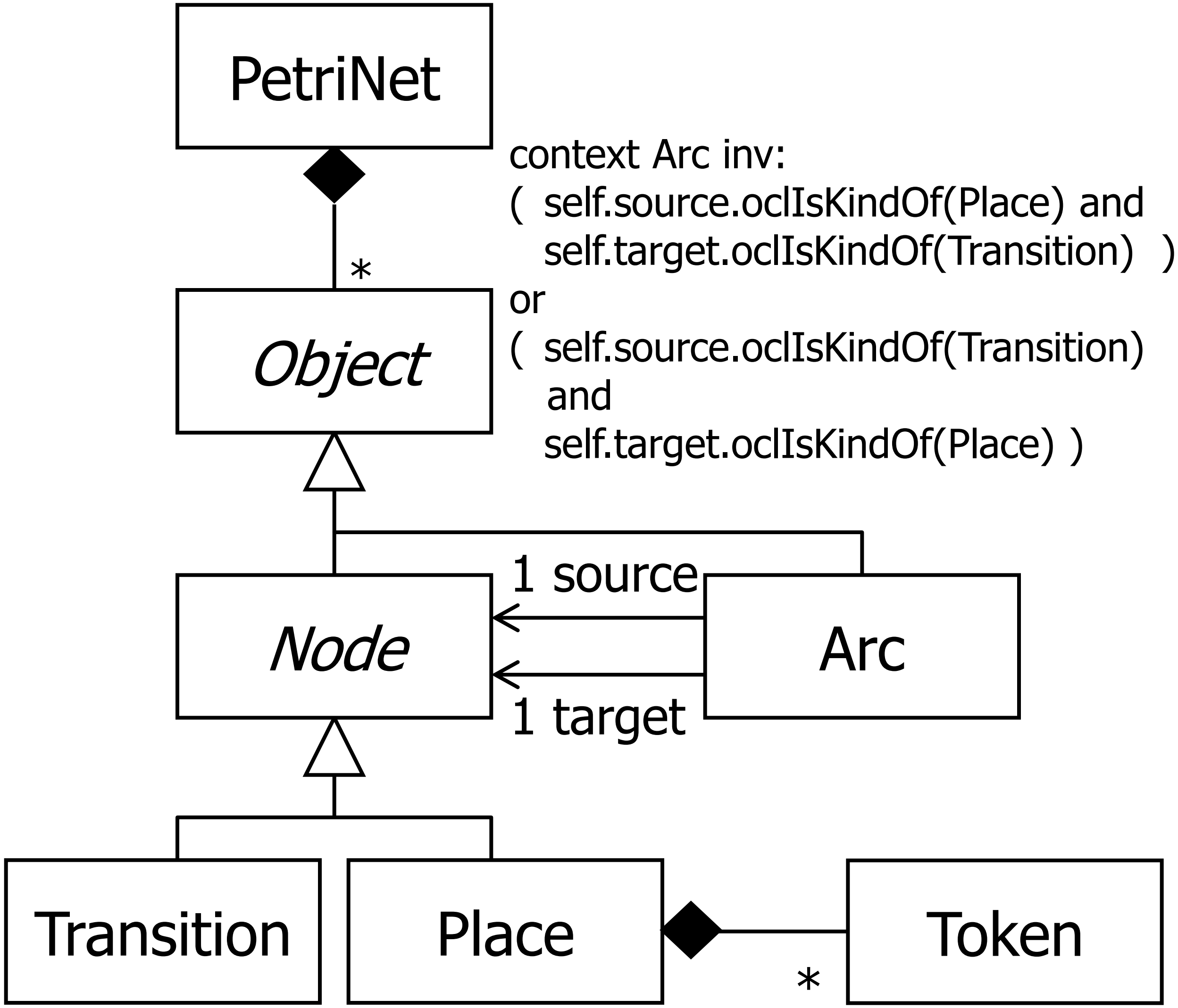
Stages

- Examples
- Taxonomy
- Glossary
- Domain model

Rule: Never ever start making a UML model without having looked at some examples first and naming the main concepts (taxonomy)!

Models (Meta Models)

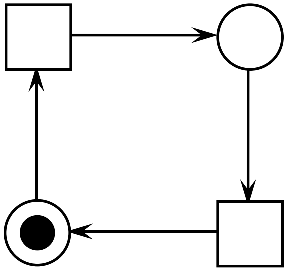




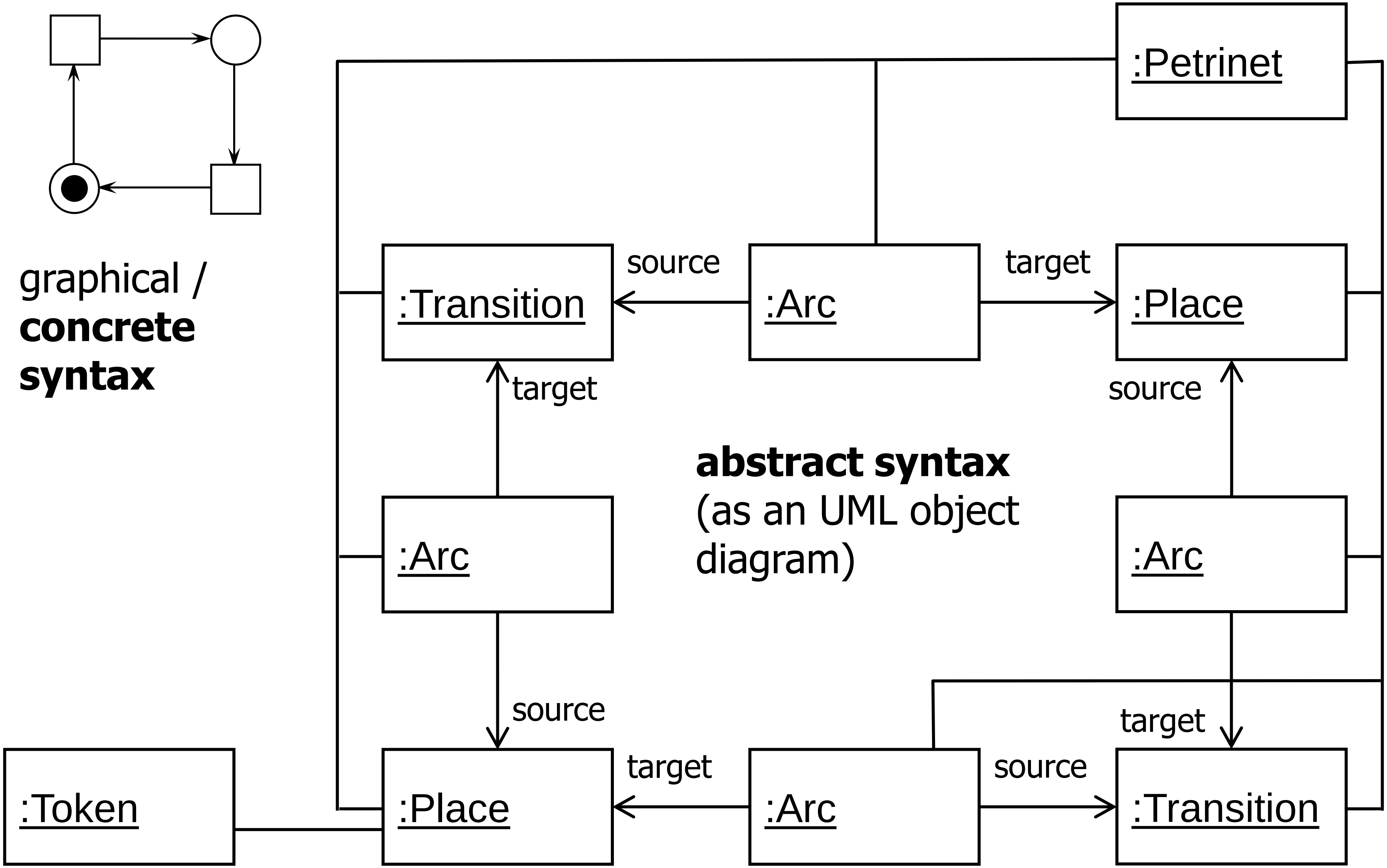
```
context Arc inv:
(  self.source.oclIsKindOf(Place) and
   self.target.oclIsKindOf(Transition)  )
or
(  self.source.oclIsKindOf(Transition)
   and
   self.target.oclIsKindOf(Place) )
```

Meta model for Petri nets

Syntax (abstract and concrete)



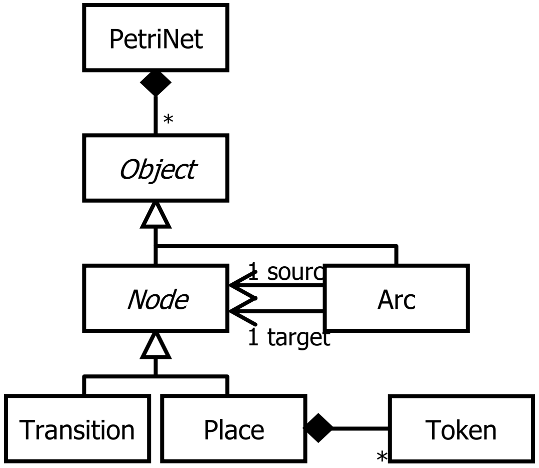
graphical / **concrete syntax**



abstract syntax (as an UML object diagram)

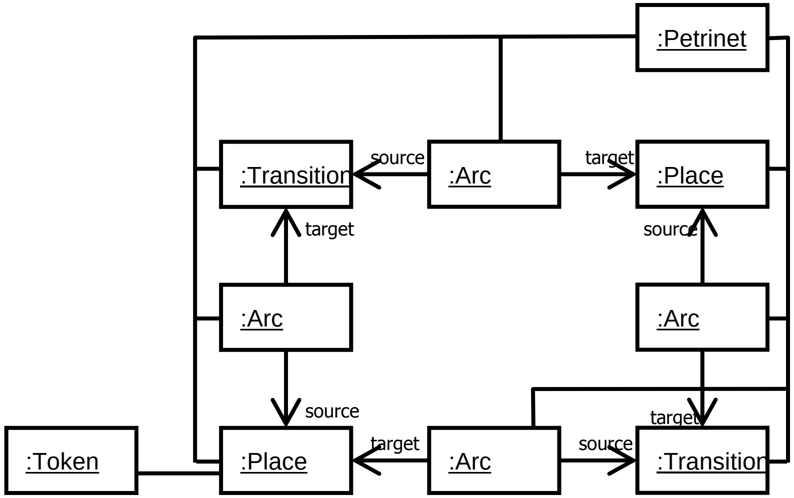
Overview

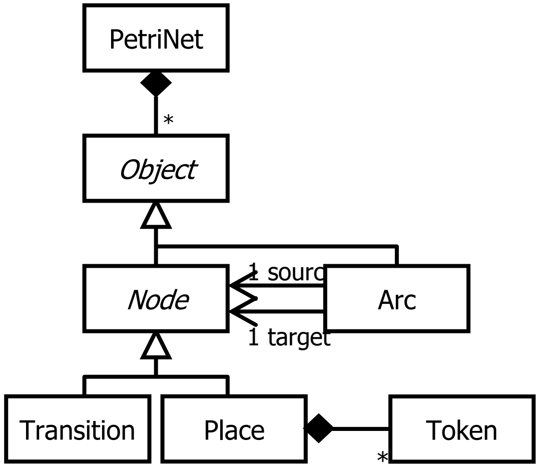
meta model build-time

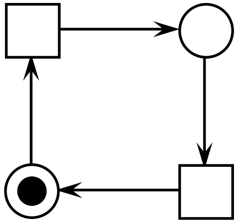


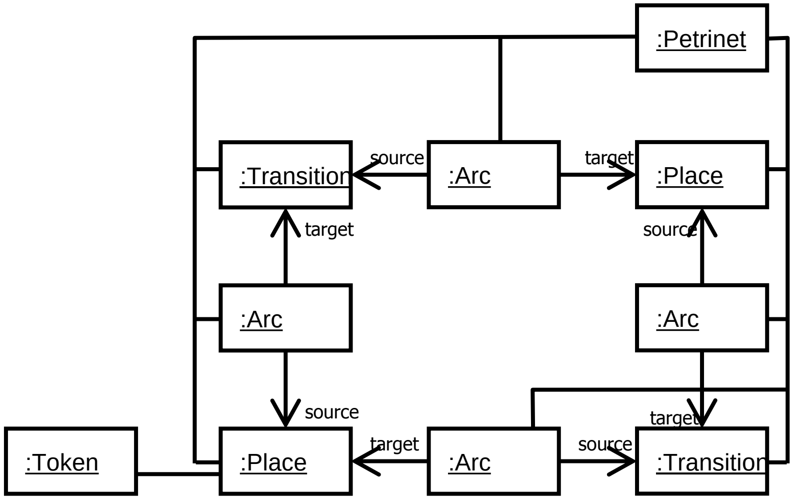
is an instance of

model runtime




PetriNet
*
Object
1 sourc
Node
Arc
1 target
Transition
Place
Token
*




:Petrinet
:Transition
source
:Arc
target
:Place
target
source
:Arc
:Arc
source
target
:Token
:Place
target
:Arc
source
:Transitior

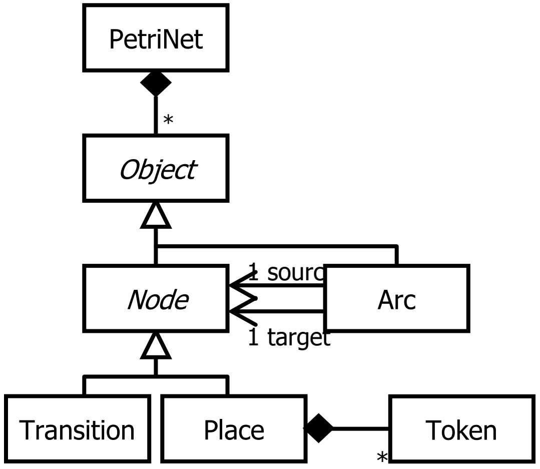
Meta modelling

- Next, we will do for class diagrams what we did for Petri nets before

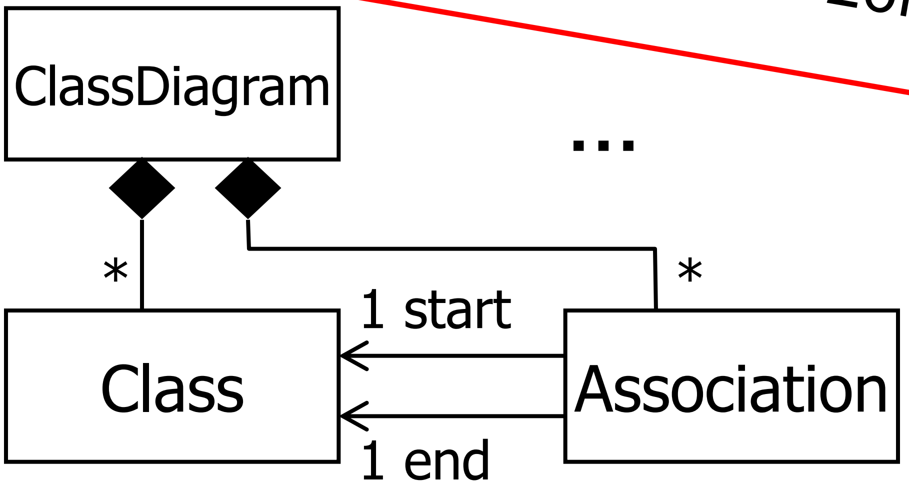
- Model for class diagrams → **meta model**

Class Diagrams are models too

Note: We will see later that this is **much more** involved (slide 26ff)!



UML model

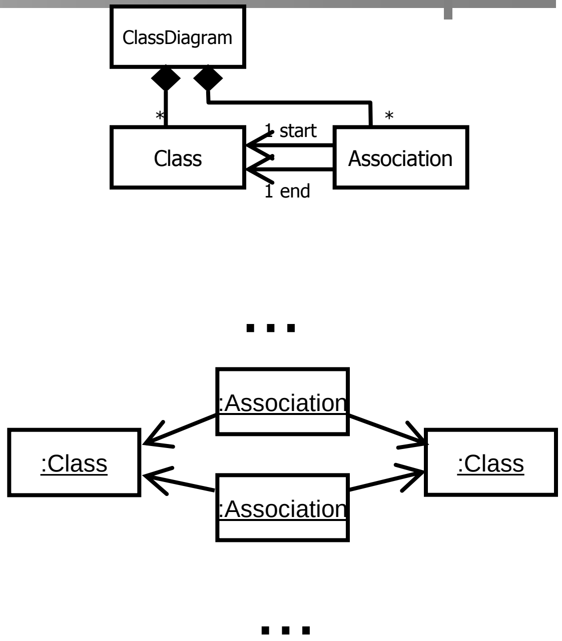


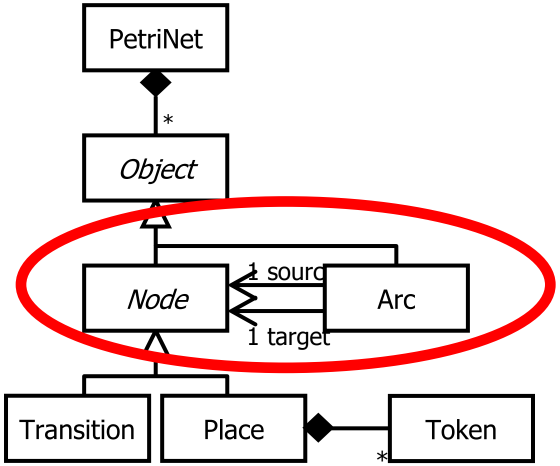
Meta-model for UML
(class diagrams)

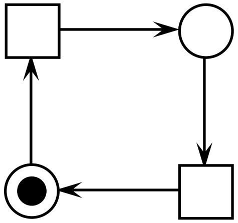
Now, the term “meta” model makes sense!

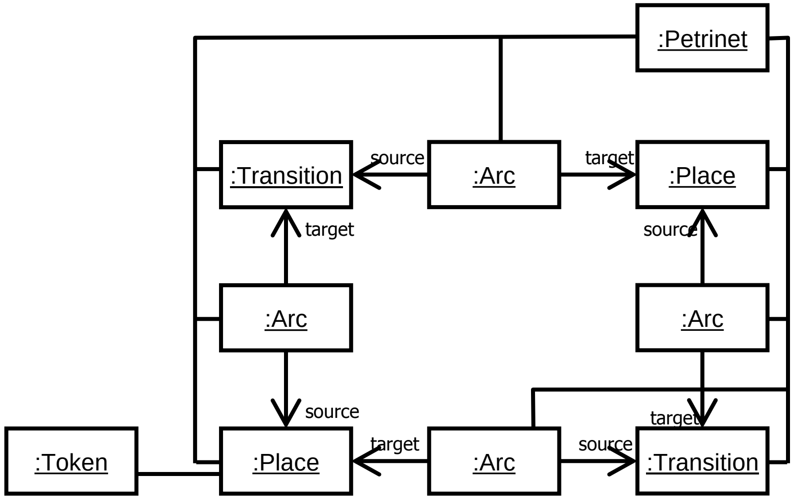
Meta-levels

We come back to that, when we learn more about the Meta Object Facility (MOF)





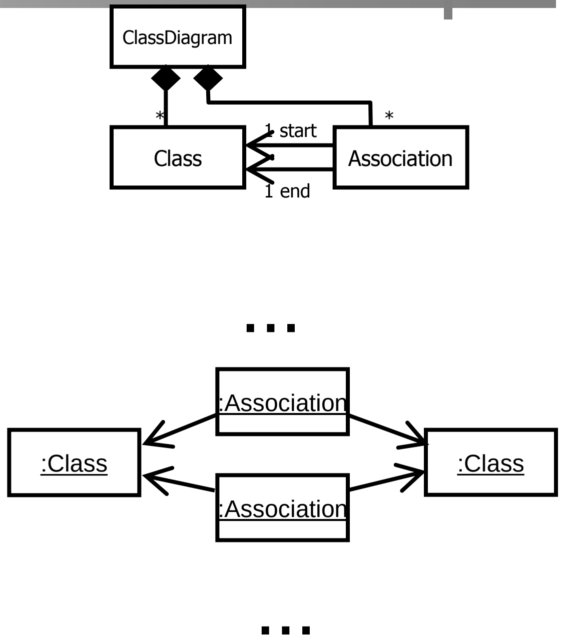


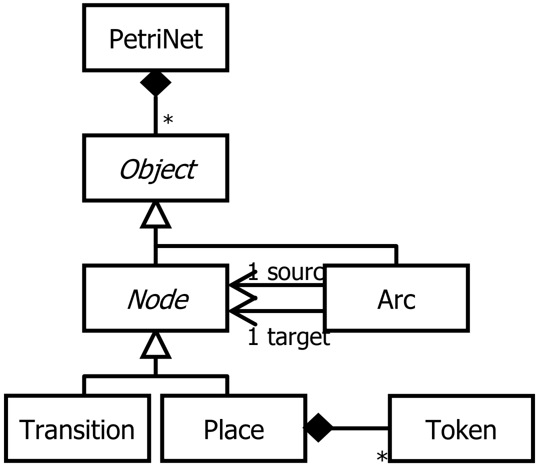


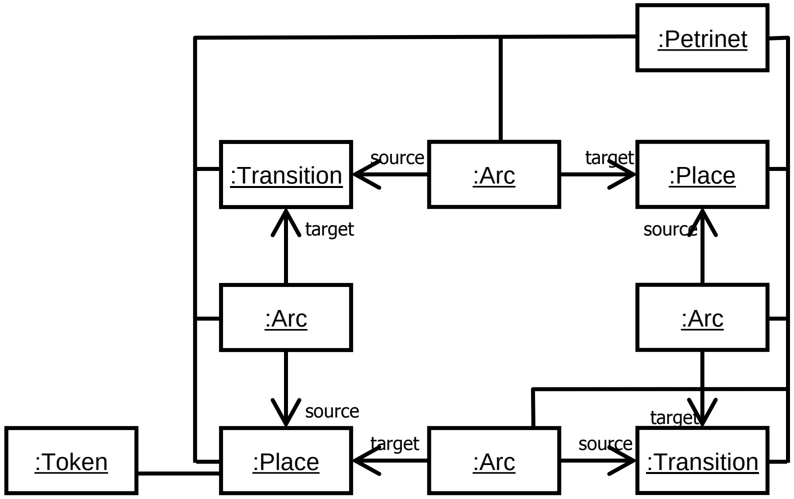
Levels of models

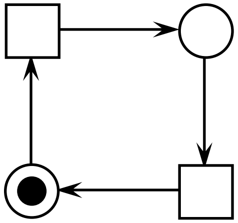
is an
instance of

concrete syntax
reprs. for









Meta-modelling (and MOF)

Meta-modelling is a core part of DSL design: Defining the abstract syntax

1. Background / Motivation

A bit coarse and “rosy” look at history!

Mid / end 90ties:

- CASE (Computer Aided Software Engineering) modelling tools become more popular
  - code generation and round-trip-engineering
  - ”UML-like” notations (and others ”Booch”, ”OMG”)
  - many dialects, variations, extensions

- Though UML starts prevailing, many other notations are in use (today called Domain Specific Languages/DSLs)

- Different ways in which code is generated

- Tools programmed manually

The technology supported by the tool was not used for its implementation.

⇒ Tools, models, generated code, ... incompatible

⇒ Hinder industrial success

Approaches

- Standardisation of a single notation: UML

- Standardisation of a transfer format

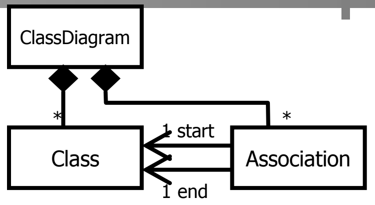
⇒Still many problems with exchanging models
⇒Need for other modeling notations

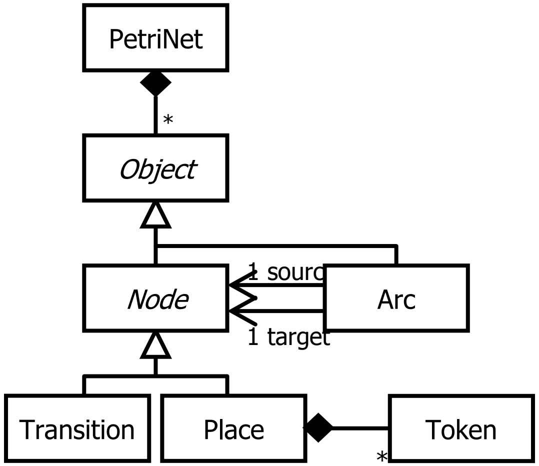
- Observation: Basic infrastructure for any CASE tool is independent from the modeling notation
- CASE tools should be implemented using their own technology

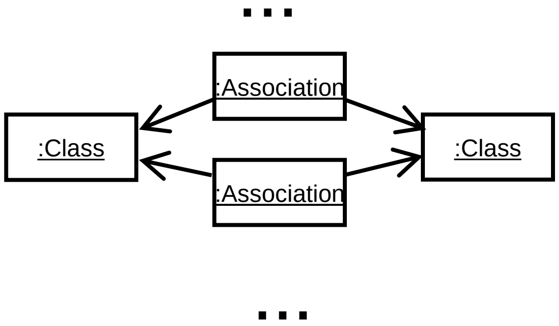
Outset

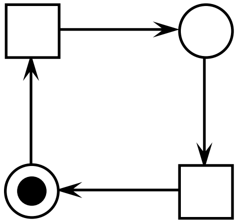
M2	Unified Modelling Notation	modelling notation
M1	Model	design-time
M0	User data	run-time

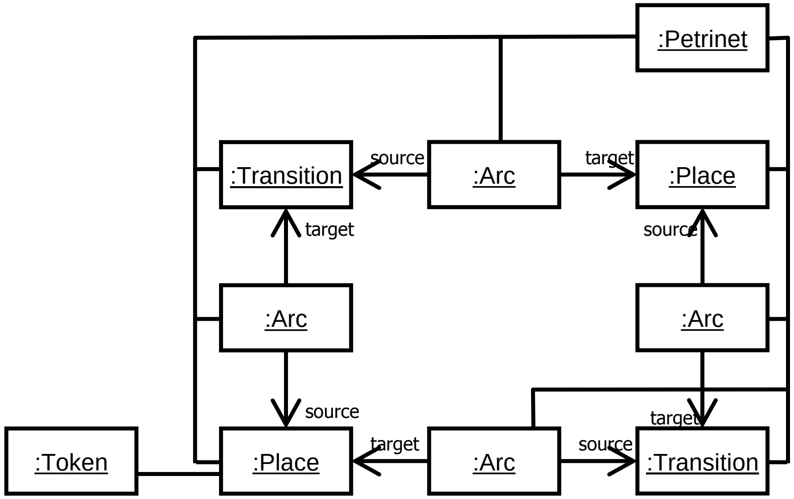
Petri net Example









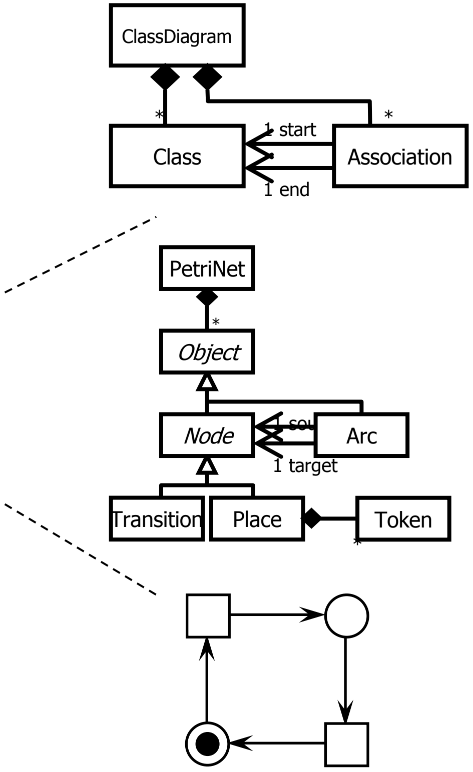


Outset

What, if we would like to use another modelling notation!

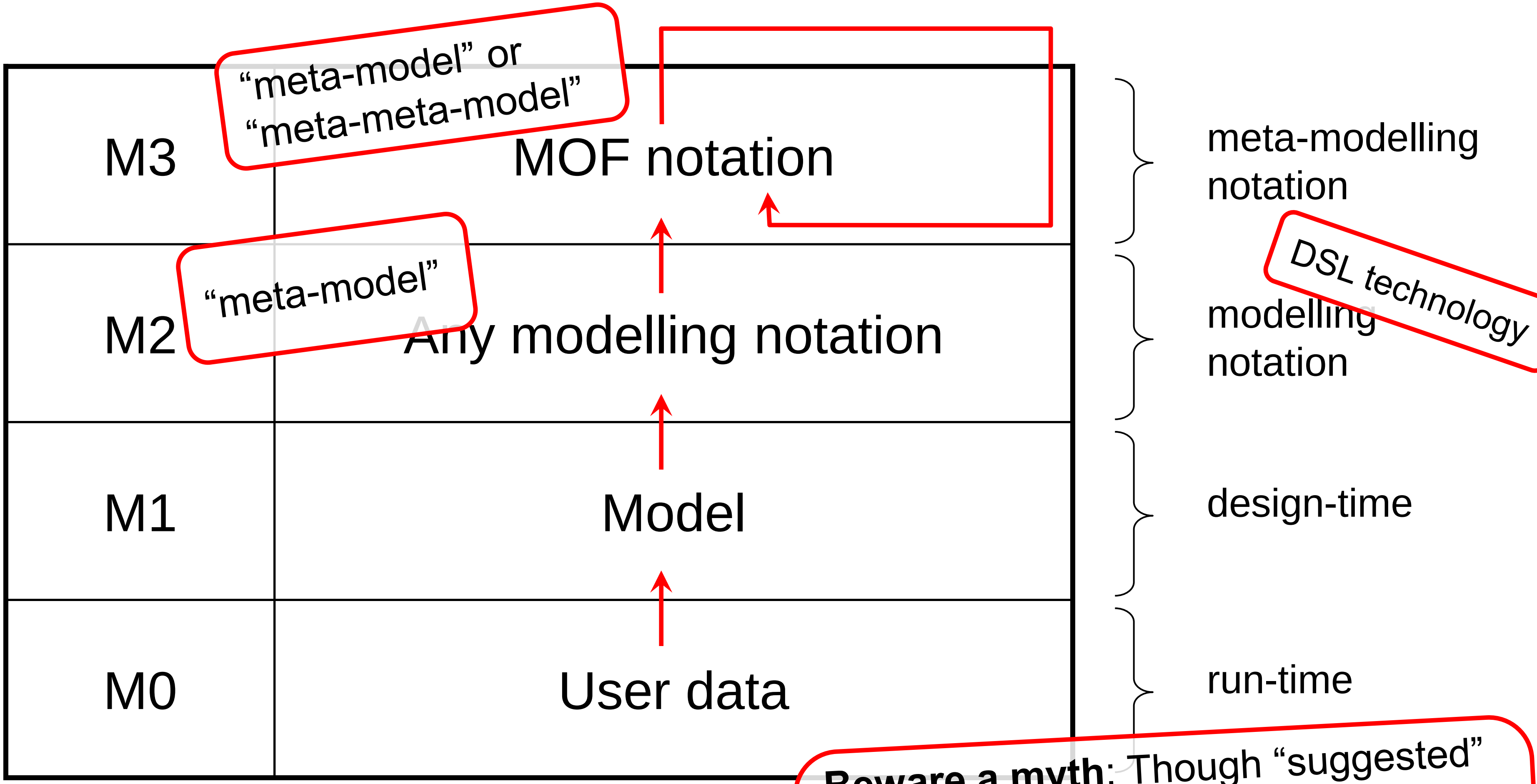
Remember: CASE tool developers should use their own medicine.

M2	Unified Modelling Notation
M1	Model
M0	User data



↑ = conforms to / is instance of

Meta Object Facility (MOF)



“meta-model” or
“meta-meta-model”

M3 MOF notation

“meta-model”

M2 Any modelling notation

M1 Model

M0 User data

meta-modelling notation

modelling notation

DSL technology

design-time

run-time

↑ = conforms to / is instance of

Beware a myth: Though “suggested” by the first versions of MOF and related standards, the number of levels is NOT fixed!
There can be any number of levels!

Meta Object Facility (MOF)

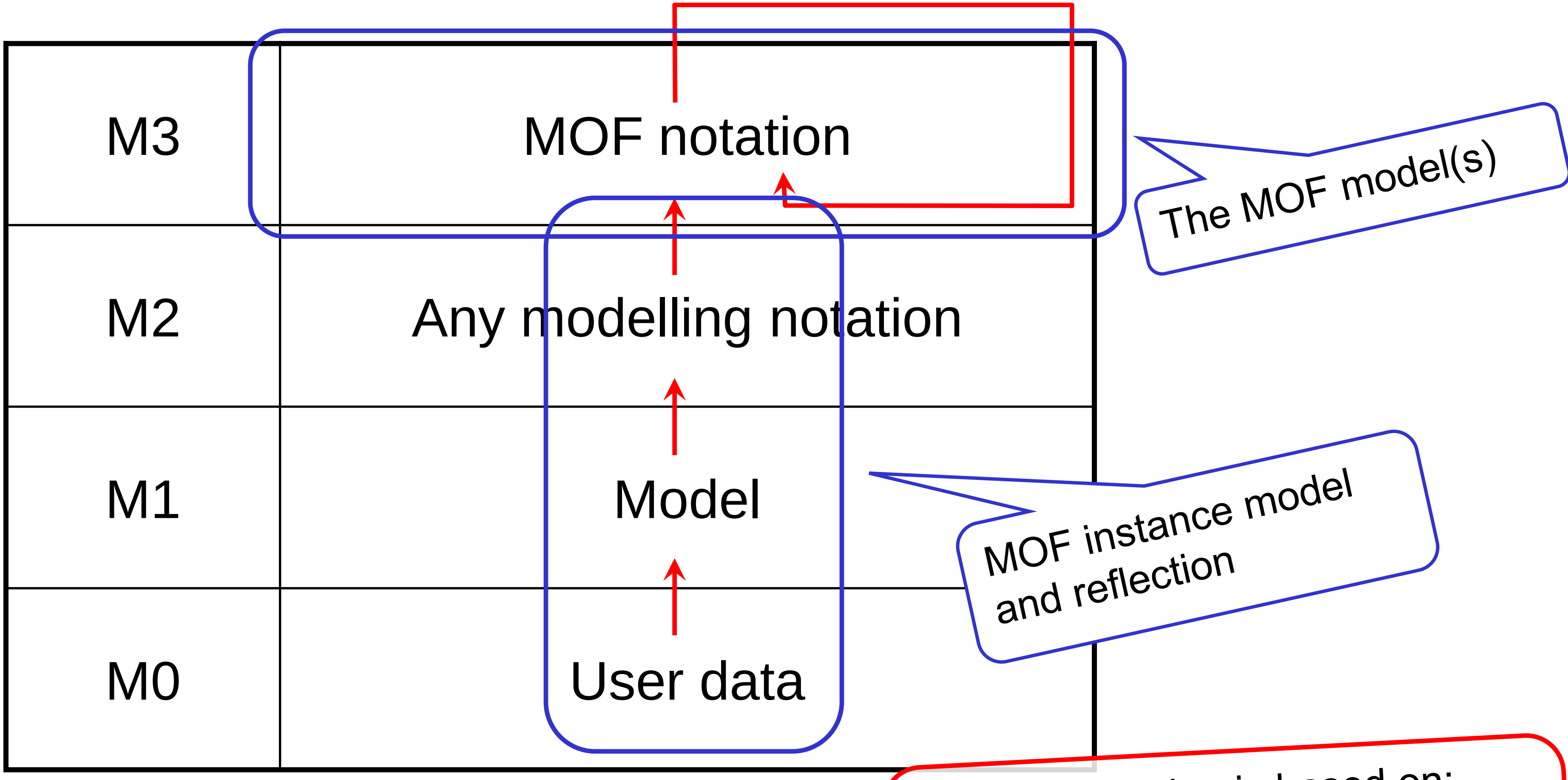
M3	MOF notation	= ECore
M2	Any modelling notation	Meta model of YAWL
M1	Model	YAWL model
M0	User data	One simulation running on a YAWL model

Discussion

- Are the four MOF levels any good?
  - There is one level that we did not have before! So, this seems to be more complicated!
  - If UML can be defined in terms of itself, why should we define it in terms of something else?

MOF distils the essence!

3. The Meta Object Facility



M3	MOF notation
M2	Any modelling notation
M1	Model
M0	User data

The MOF model(s)

MOF instance model and reflection

↑ = conforms to / is instance of

This presentation is based on: Meta Object Facility (MOF) Core Specification, Version 2.0, OMG formal/06-01-01

Meaning of ”Meta-”

Meta (from Greek: μετά = "after", "beyond", "with", "adjacent", "self"), is a prefix used in English in order to indicate a concept which is an abstraction from another concept, used to complete or add to the latter.

In epistemology, the prefix **meta-** is used to mean *about (its own category)*. For example, metadata are data about data, something about something (who has produced them, when, what format the data are in and so on). Similarly, metamemory in psychology means an individual's knowledge about whether or not they would remember something if they concentrated on recalling it. Furthermore, metaemotion in psychology means an individual's emotion about his/her own basic emotion, or somebody else's basic emotion.

Another, slightly different interpretation of this term is "about" but not "on" (exactly its own category). For example, in linguistics a grammar is considered as being expressed in a metalanguage, or a sort of language for describing *another* language (and not itself). A **meta-answer** is not a real answer but a reply, such as: "*this is not a good question*", "*I suggest you ask your professor*". Here, we have such concepts as meta-reasoning and meta-knowledge.

…

From: http://en.wikipedia.org/wiki/Meta

Meaning of meta

Co-notations and meaning in Software Engineering:

- beyond, ”one level higher”
- possibly self-referential
  (with all the problems of self-referentiality)

Self-references are at the core of all paradoxes.
Example: “This statement is wrong”!

Often also:

- ~~a UML model~~
- ~~a class diagram~~

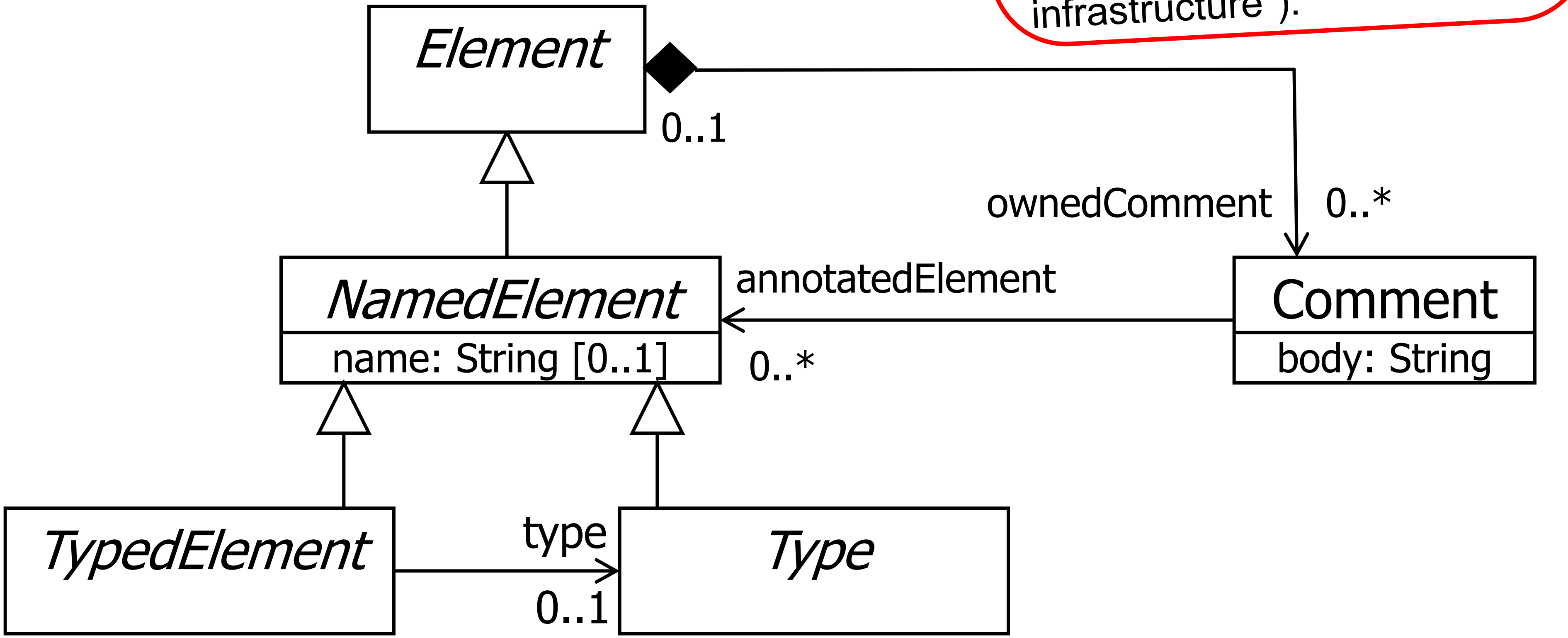
Abuse of language introduced by people working only or just too much on the meta-level.

3.1. EMOF model

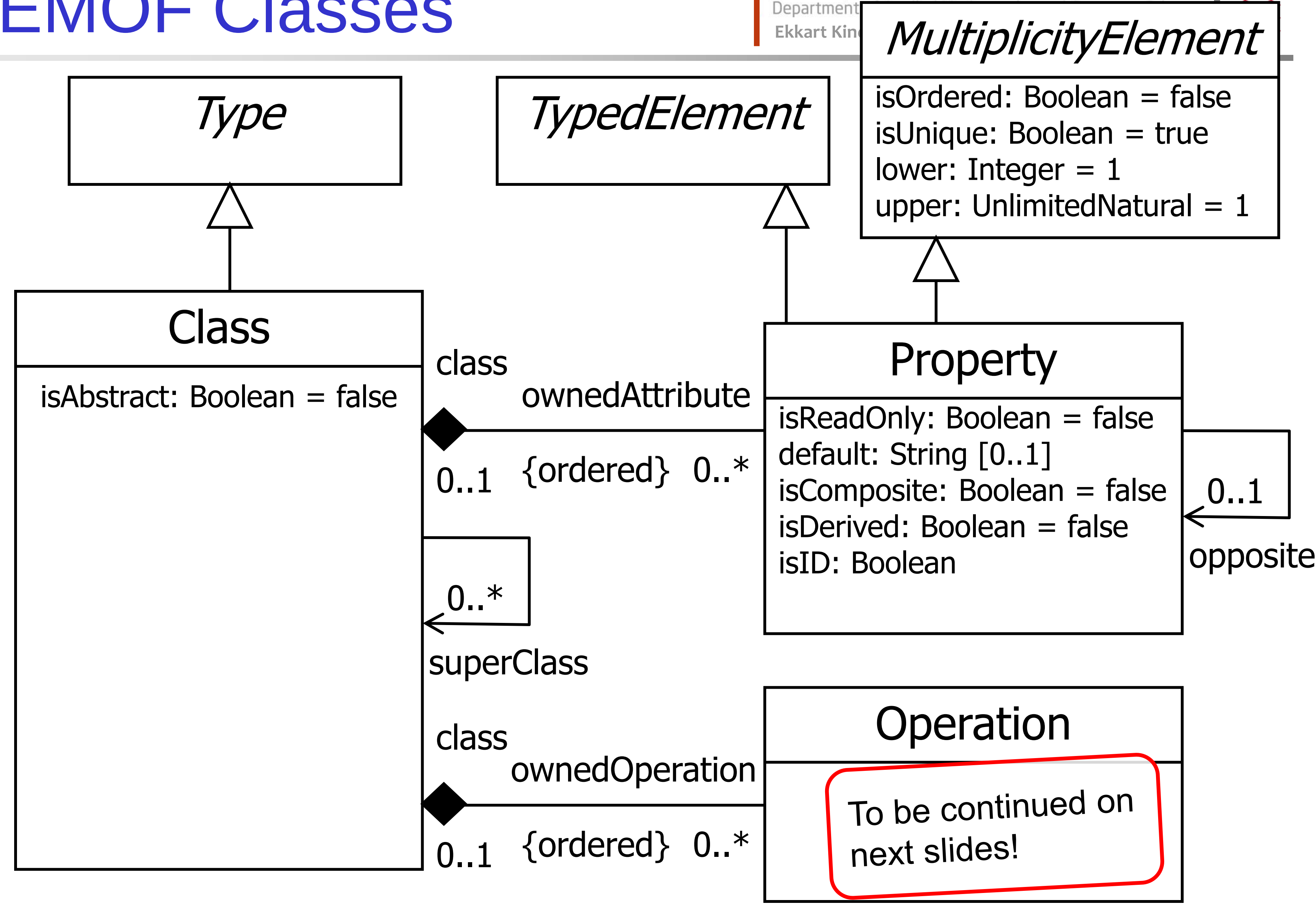
Actually, this comes from the UML infrastructure Core::Basic.

The MOF standard refers to and uses concepts and notations from the UML standard (the “UML infrastructure”).

EMOF Types



EMOF Classes



To be continued on next slides!

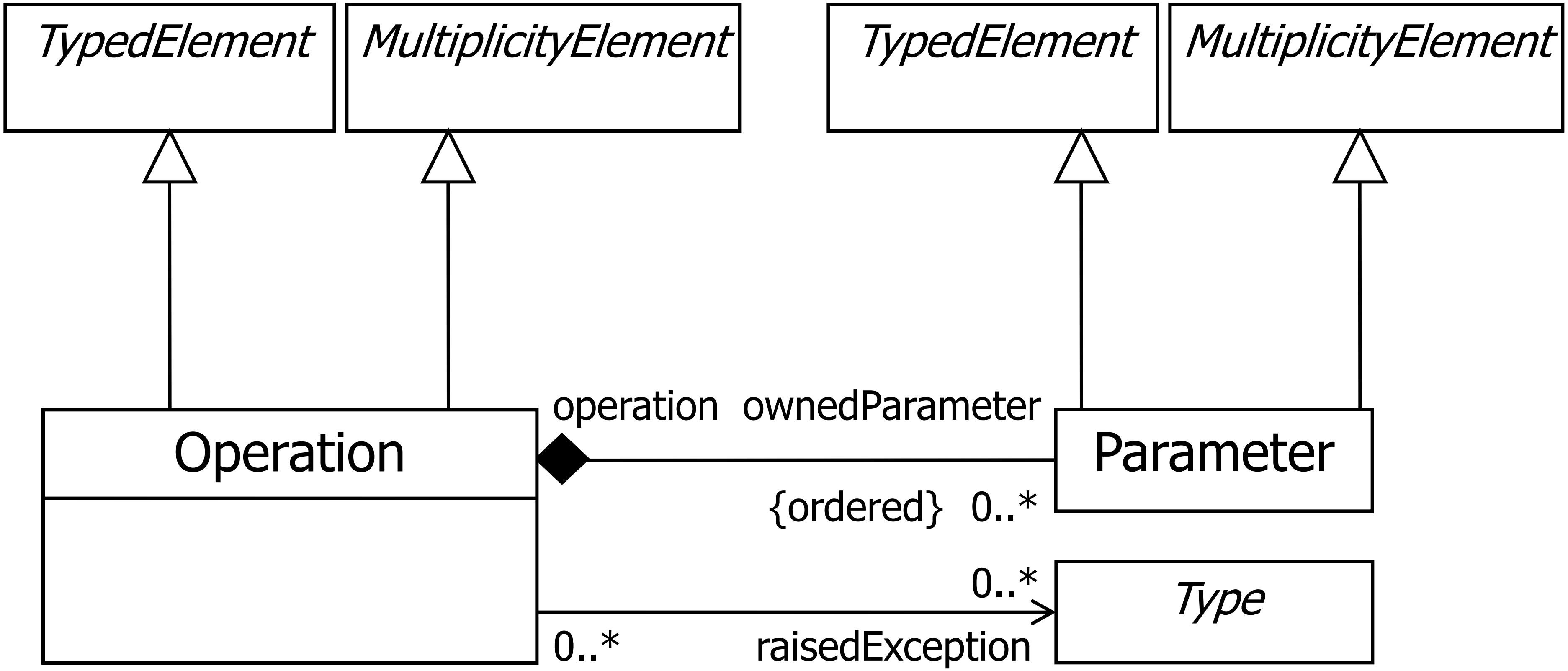
EMOF Classes (cntd.)

Additional constraints (e.g.):

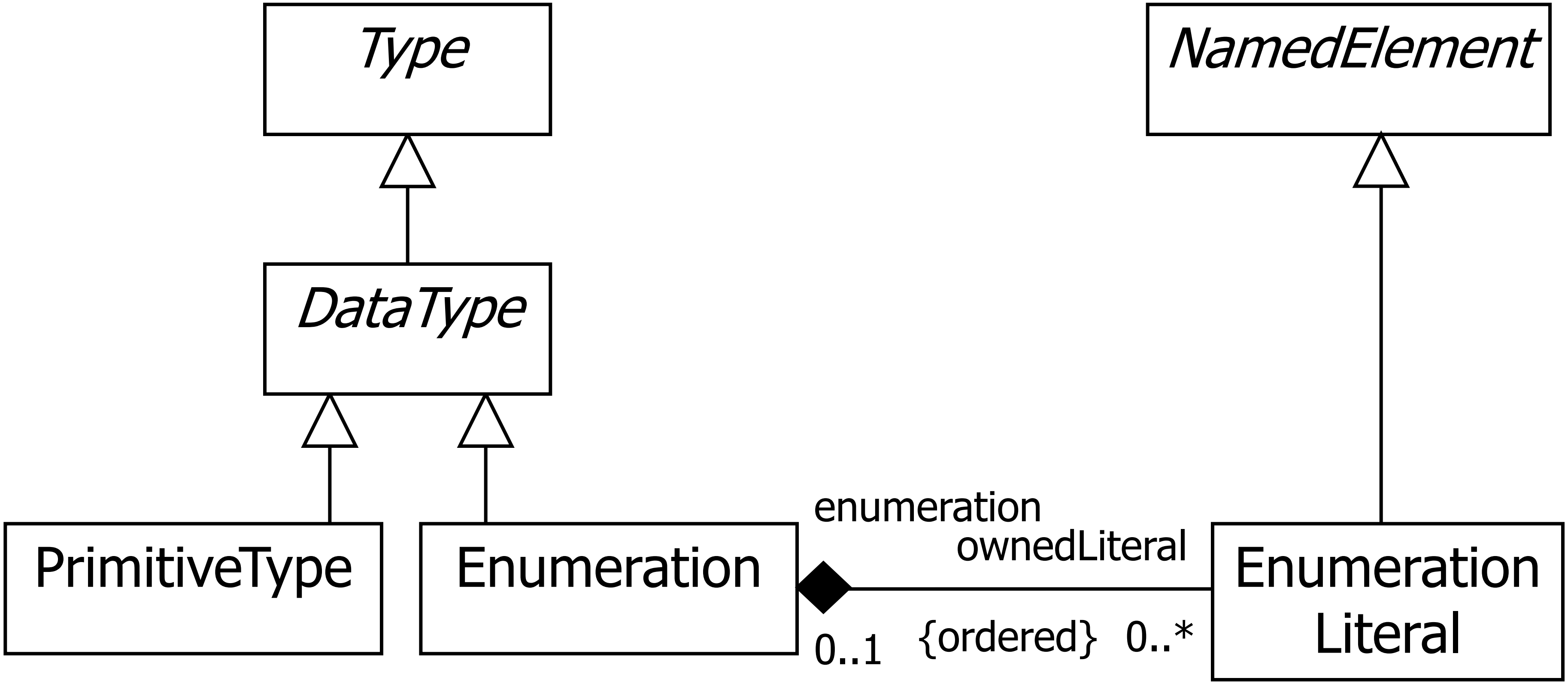
- opposite properties are properly paired
- no cycles in inheritance structure
- an object can be contained in at most one container

EMOF Classes (cntd.)

Details of slides 29 – 35 not too relevant for this course!



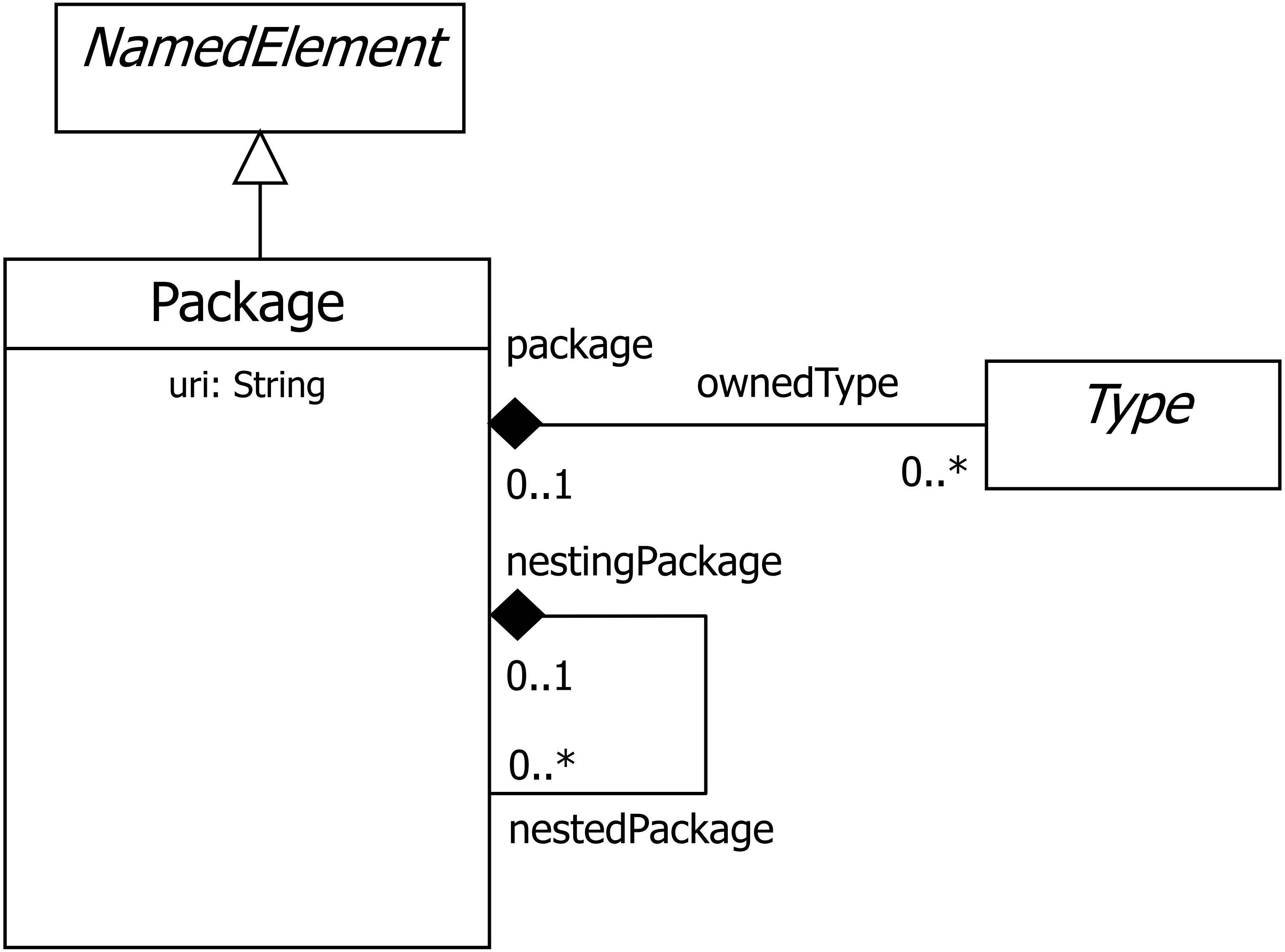
EMOF Data Types



EMOF Primitive Types

- Boolean
- String
- Integer
- UnlimitedNatural (* for ”infinity”)

EMOF Packages



EMOF Discussion

- Can EMOF be defined with its own concepts?
- Is it expessible enough?
- What is missing (as compared to UML diagrams)?
- How does EMOF relate to ECore (the model underlying EMF)?
- Can UML be expressed in it?
- Any other problems?

EMOF stands for Essential MOF; we will discuss more complete model, Complete MOF (CMOF), later.

EMF / ECore might be the reason, EMOF was included in the MOF standard.

3.2 Reflection capability

- Creating models and their instances (resp. meta-models and their conforming models) dynamically

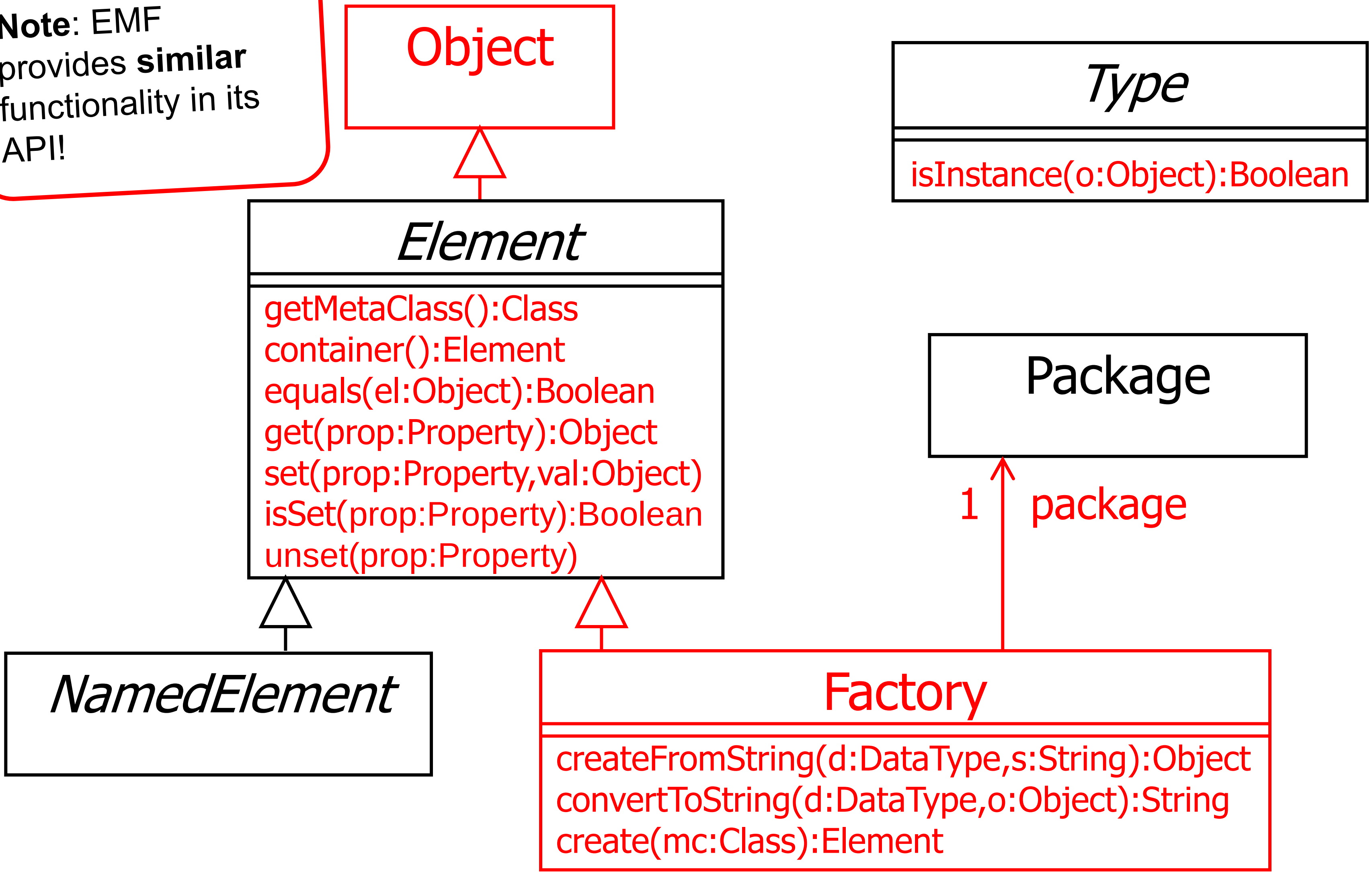
- Navigating between model elements and instance

⇒By navigation between different meta-levels in an arbitrary way, MOF is not restricted to a fixed number of levels.

"Reflection": Knowing something (and reasoning) about oneself.

Reflection package

Note: EMF provides **similar** functionality in its API!


Object
Type
isInstance(o:Object):Boolean
Element
getMetaClass():Class
container():Element
equals(el:Object):Boolean
get(prop:Property):Object
set(prop:Property,val:Object)
isSet(prop:Property):Boolean
unset(prop:Property)
Package
1
package
NamedElement
Factory
createFromString(d:DataType,s:String):Object
convertToString(d:DataType,o:Object):String
create(mc:Class):Element

For properties with more than one value, there exist ReflexiveCollection and ReflexiveSequence (similar to Java Collections)!

4. XMI

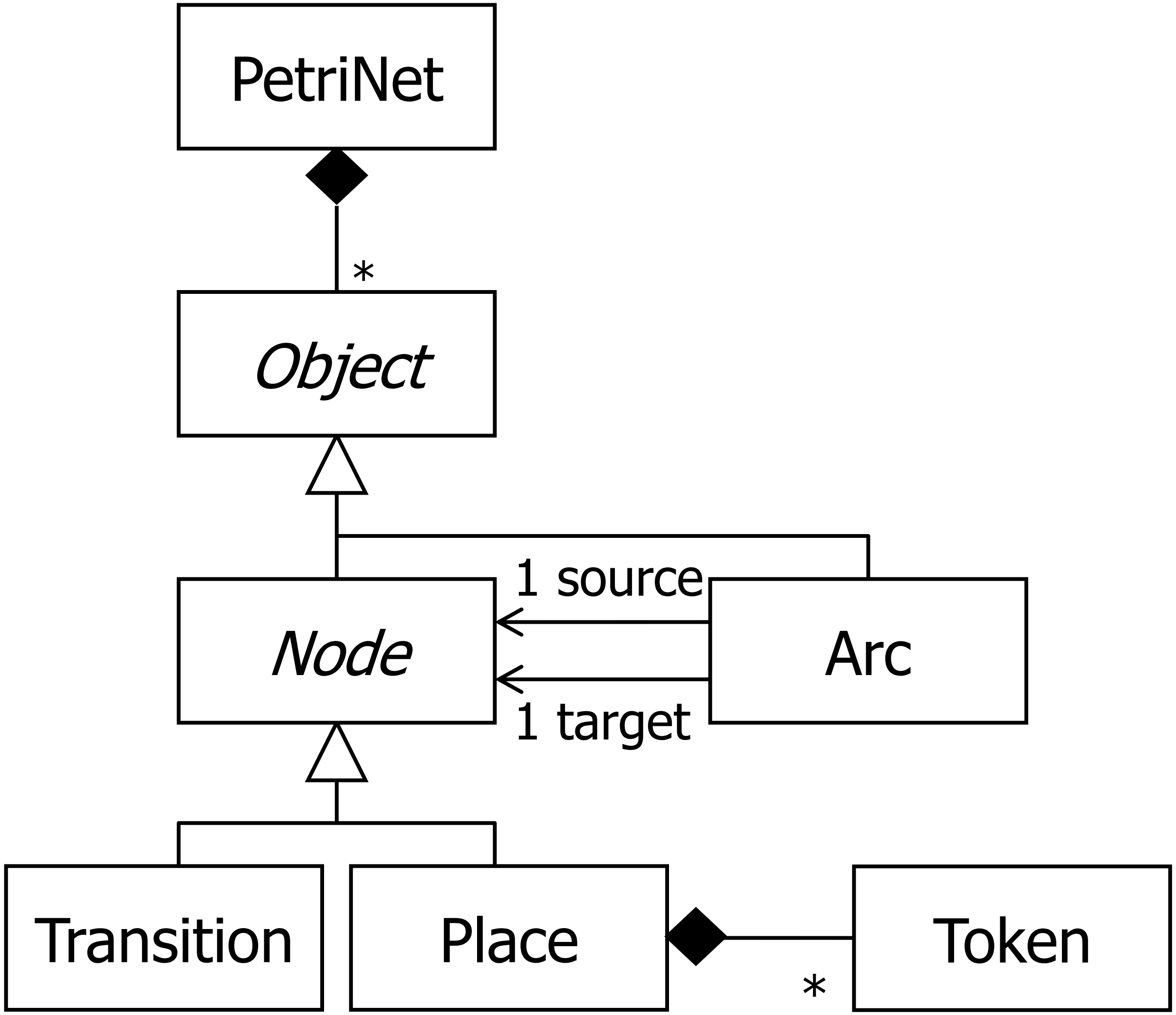
- Mapping MOF-models and its instances to XML in a standard way
- A MOF model is mapped to an XMLSchema for its instances
- XMI is a standard associated with MOF

=>You can easily exchange MOF models

=>Once you agree on the MOF-model, you can exchange instances of that model

Warning: If you change the meta model, you often can no longer read older versions of XMI instances of it! That is why XML syntax is often explicitly defined.

Example (model)



Meta model for Petri nets

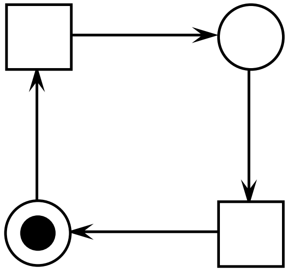
Example: EMOF model

```
<?xml version="1.0" encoding="UTF-8"?>
<emof:Package xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:emof="http://schema.omg.org/spec/MOF/2.0/emof.xml" xmi:id="PetriNets"
    name="PetriNets" uri="APetriNetEditorIn15Minutes">
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.PetriNet" name="PetriNet">
    <ownedAttribute xmi:id="PetriNets.PetriNet.object" name="object"
        isOrdered="true"
        lower="0" upper="*" type="PetriNets.Object" isComposite="true"/>
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Object" name="Object"
      isAbstract="true"/>
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Node" name="Node"
      isAbstract="true„ superClass="PetriNets.Object">
    <ownedAttribute xmi:id="PetriNets.Node.name" name="name" isOrdered="true"
        lower="0">
      <type xmi:type="emof:PrimitiveType"
          href="http://schema.omg.org/spec/MOF/2.0/emof.xml#String"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="PetriNets.Node.in" name="in" isOrdered="true"
        lower="0" upper="*" type="PetriNets.Arc"
        opposite="PetriNets.Arc.target"/>
    <ownedAttribute xmi:id="PetriNets.Node.out" name="out" isOrdered="true"
        lower="0" upper="*" type="PetriNets.Arc"
        opposite="PetriNets.Arc.source"/>
  </ownedType>
```

Example (cntd.)

```
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Arc" name="Arc"
        superClass="PetriNets.Object">
    <ownedAttribute xmi:id="PetriNets.Arc.source" name="source"
   isOrdered="true"
        type="PetriNets.Node" opposite="PetriNets.Node.out"/>
    <ownedAttribute xmi:id="PetriNets.Arc.target" name="target"
   isOrdered="true"
        type="PetriNets.Node" opposite="PetriNets.Node.in"/>
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Transition"
      name="Transition" superClass="PetriNets.Node"/>
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Place"
      name="Place" superClass="PetriNets.Node">
    <ownedAttribute xmi:id="PetriNets.Place.token" name="token"
        isOrdered="true" lower="0" upper="*„
        type="PetriNets.Token" isComposite="true"/>
  </ownedType>
  <ownedType xmi:type="emof:Class" xmi:id="PetriNets.Token"
      name="Token"/>
  <xmi:Extension extender="http://www.eclipse.org/emf/2002/Ecore">
    <nsPrefix>APetriNetEditorIn15Minutes</nsPrefix>
  </xmi:Extension>
</emof:Package>
```

Example (instance)



XMI instance

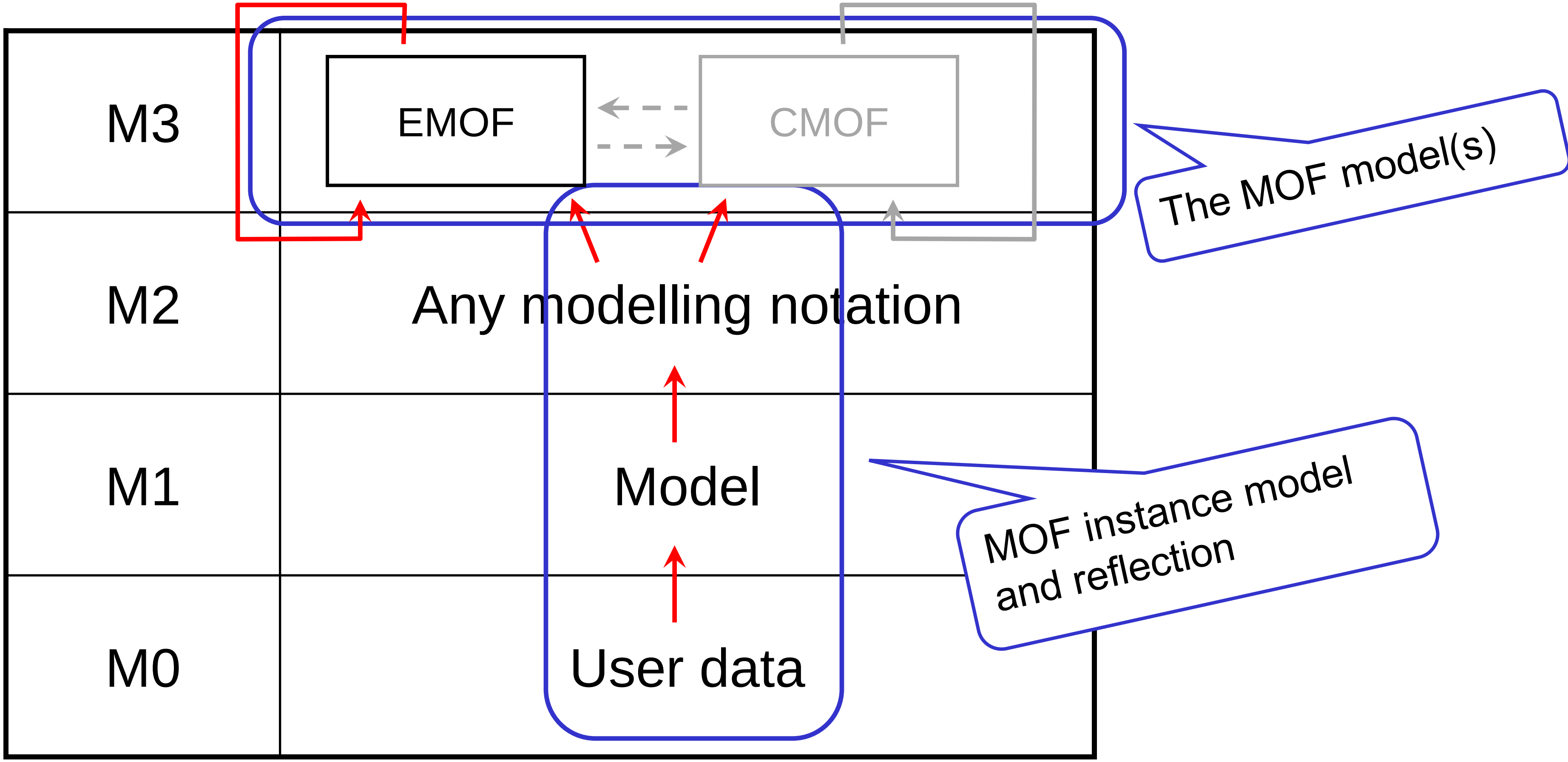
```
<?xml version="1.0" encoding="UTF-8"?>
<APetriNetEditorIn15Minutes:PetriNet xmi:version="2.0"
   xmlns:xmi="http://www.omg.org/XMI"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:APetriNetEditorIn15Minutes="APetriNetEditorIn15Minutes">
  <object xsi:type="APetriNetEditorIn15Minutes:Transition" name="t1"
     in="//@object.7" out="//@object.4"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Transition" name="t2"
     in="//@object.5" out="//@object.6"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Place" name="p1"
     in="//@object.6" out="//@object.7">
    <token/>
  </object>
  <object xsi:type="APetriNetEditorIn15Minutes:Place" name="p2"
     in="//@object.4" out="//@object.5"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Arc"
     source="//@object.0" target="//@object.3"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Arc"
     source="//@object.3" target="//@object.1"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Arc"
     source="//@object.1" target="//@object.2"/>
  <object xsi:type="APetriNetEditorIn15Minutes:Arc"
     source="//@object.2" target="//@object.0"/>
</APetriNetEditorIn15Minutes:PetriNet>
```

The serialisation of instances can be customized in tools like EMF.

Here, references are via paths (XPath).

If model elements have ids, the references will be via the ids!

6. Summary



↑ = conforms to / is instance of

Domain Specific Languages

- Domain Specific Language (DSL)
- Domain Specific Languages (DSLs)

What do they mean?
What is their “spirit”?

DSL / DSLs

- The terms DSL and DSLs are uses since the the mid 90ties; ”Domain Specific Automatic Programming” even dates back to the mid 80ties*.

*) D. R. Barstow: Domain-Specific Automatic Programming. IEEE TSE, Vol. SE-11, no. 11, Nov. 1985, pp. 1321-1336

- Still, there is not is not a uniform or universal understanding of what a DSL or what DSLs are; it depends a bit on the background which characterisitics of DSLs are considered to but important or relevant.

*DSLs and MBSE are sometimes used almost synonymously.

- This lecture gives an overview – but with a model-based software engineering bias!

- DSL (singular):
  A single domain specific language,
  designed and realised according to some principles
  and for a specific purpose or a specific domain

- DSLs (plural):
  - Disipline and principles for designing and realising a DSL
  - A technology or set of technolgies for designing and realising a DSL (mostly from MBSE)
  - A way of ”thinking” software design (idioms)

1. Examples of DSLs

- COBOL
- Lisp
- PROLOG

Some examples named by some proponents of DSLs; not all would agree that these are DSLs!

- SQL (Structured Query Language → DB)
- BNF (Backus Naur Form → syntax definition)
- regex (regular expressions)
- lex, yacc (compiler construction)

- Shell scripting languages
- OCL

Some DSL existed even before the term DSL was invented!

- BPEL (Business process execution language)
  BPML (Business process modeling language)
- **YAWL**

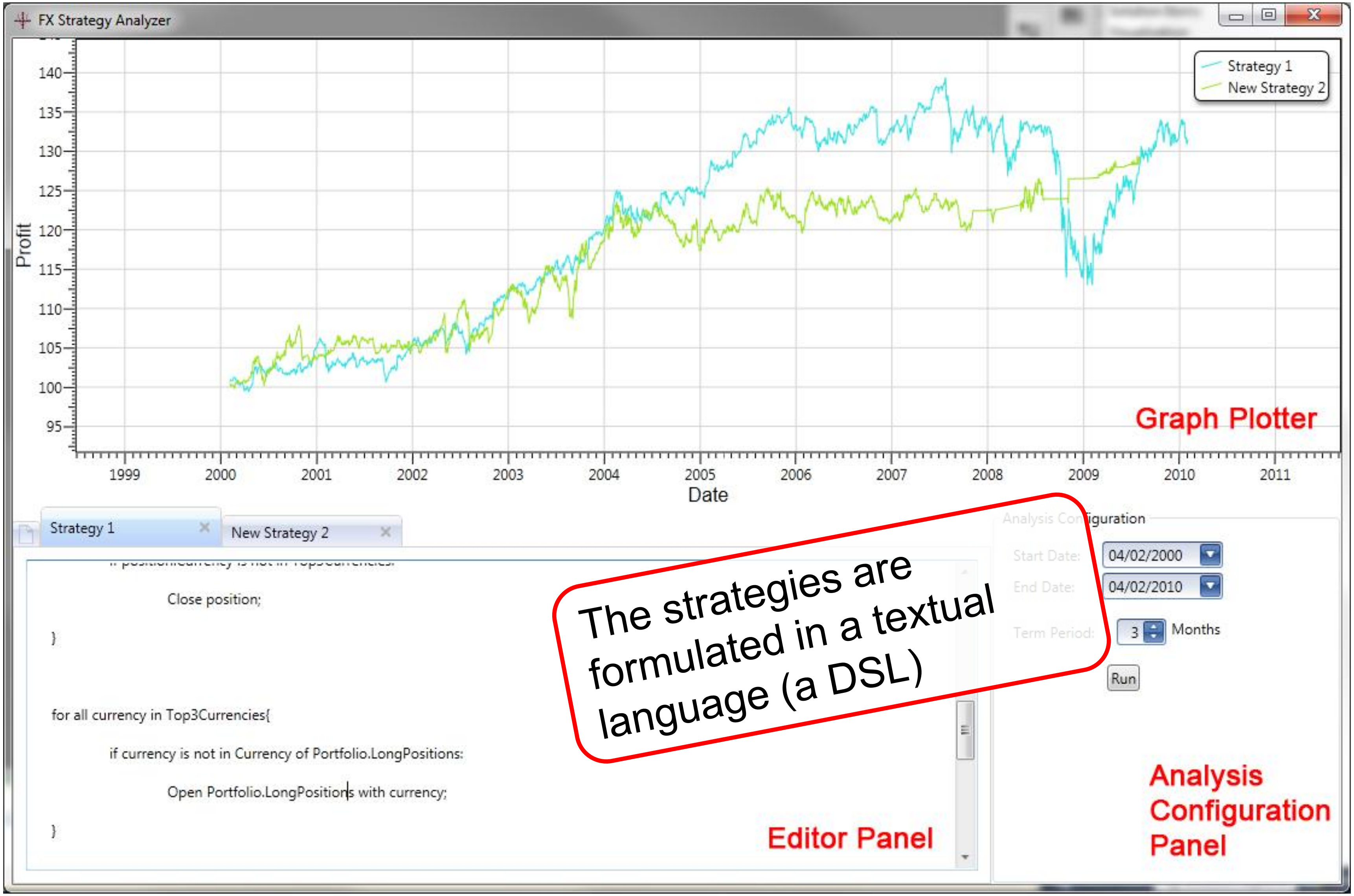
- Petri nets
- ECNO

It is debatable whether Petri nets and ECNO are actually DSL

- **Trading strategy language (see next slide)**

- PDF / PostScript
- HTML / CSS

Tool for testing FX strategies



```
        if positionCurrency is not in Top3Currencies:
                Close position;
}

for all currency in Top3Currencies{
        if currency is not in Currency of Portfolio.LongPositions:
                Open Portfolio.LongPositions with currency;
}
```

Analysis Configuration

Start Date: 04/02/2000

End Date: 04/02/2010

Term Period: 3 Months

Graph Plotter

The strategies are formulated in a textual language (a DSL)

Editor Panel

Analysis Configuration Panel

Counter-examples

- C
- C++
- C#
- Java
- Ruby
- Scala
- ...

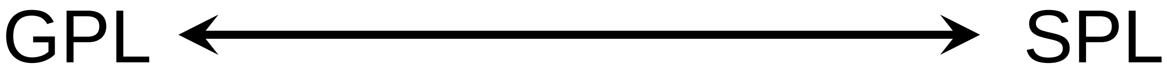
- UML
- ...

2. Characteristics

Traditional distinction of ”programming languages”:

- General Purpose Languages (GPL):
  - universal
  - The same thing can be achieved in many differnent ways
  - Turing complete (can compute everthing)
  - huge
- Special Purpose Language (SPL):
  - made for a specific purpose
    (adequate for this specific purpose)
  - succinct and highly expressive (for a given purpose)
  - typically, not Turing complete
  - small

SPL and DSL?



GPL

SPL

- Is any SPL a DSL?
- Is every DSL a SPL?

DSL characteristics

- Textual (language) vs graphical (notation)
- Programming vs. **modelling**
- **Domain of application** vs separation of concerns
- Way of thinking design vs
  use of specific DSL technologies
- **Abstraction** vs technical
- **User focus** vs technical focus
- Language vs framework
- **Idiom oriented** vs. programming oriented

Classification

- **Embedded DSL**:
  Embedded to an existing programming language by adding some framework for some purpose (often some functional languages with syntactic sugaring features)

- Typically textual languages!
- Often programmed (with ”DSL thinking” in mind)

- **External DSL**:
  Standalone language (graphical/textual) which is then compiled or interpreted. Often realized by DSL development tech

- Often: Focus on adequate concrete syntax!
- Typically realized by using ”DSL technologies”

3. Parts of a DSL definition

- **Abstract syntax** (see L01):
  language concepts and their relation
  (API / domain model / framework)

- **Concrete syntax** (see L01):
  syntactical representation of concepts
  (graphical or textual)

Actually, there could be different concrete syntax for the same abstract syntax

- **Semantics** (what it does):
  Code generation or interpretation, which enacts what an instance of the DSL says

DSL Technolgies typically support the first two steps; and might help a bit with the last!

4. Philosophy

- A DSLs should help decrease redundancy and unnecessary work
- A DSL should help separating the variable or generic parts of a software product from parts which do not change
- A DSL should increase reuse
- A DSL should support abstraction form irrelevant technical details
- A DSL should emphasize the domains idioms

Discussion

- Are Petri nets a DSL?

- To which extent is the course's project (YAWL editor/simulator) a ”DSL” or ”DSLs”

Discussion

- MBSE Technologies help implementing DSLs fast and efficiently (mostly concerning abstract and concrete syntax)

- Therefore, the terms MBSE and DSL are often used in the same context (and sometimes mixed up)