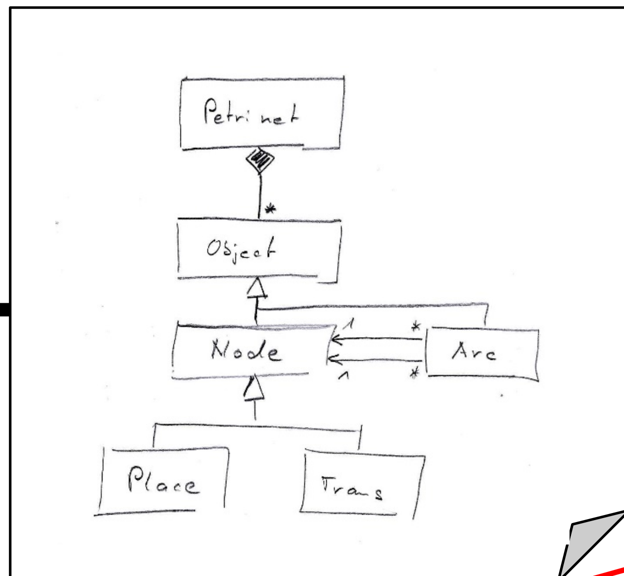
The ePNK: A model bases development project

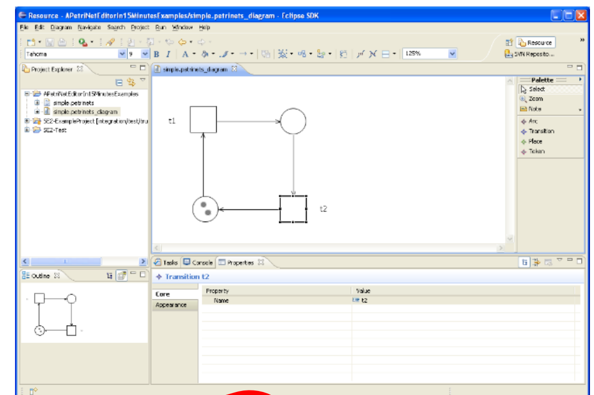
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

Motivation

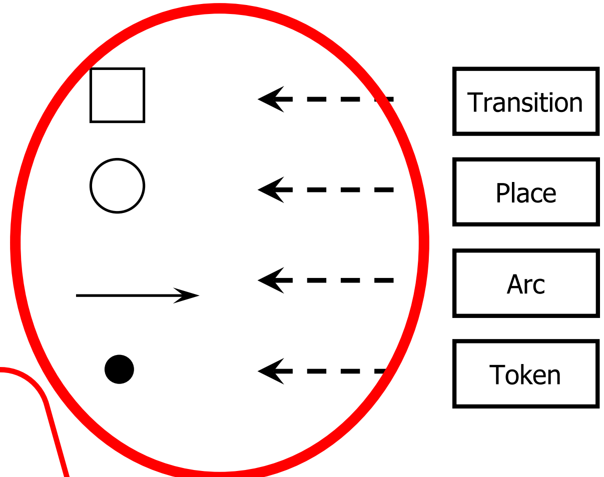
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

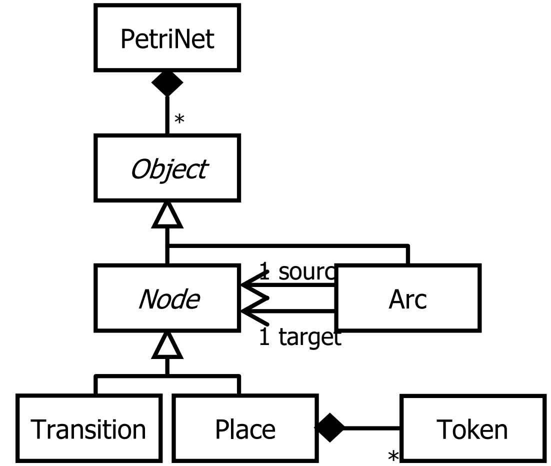




"A Petri net editor in 15 minutes"

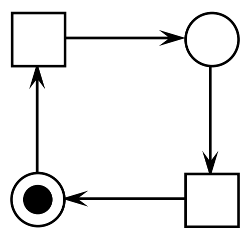
Motivation

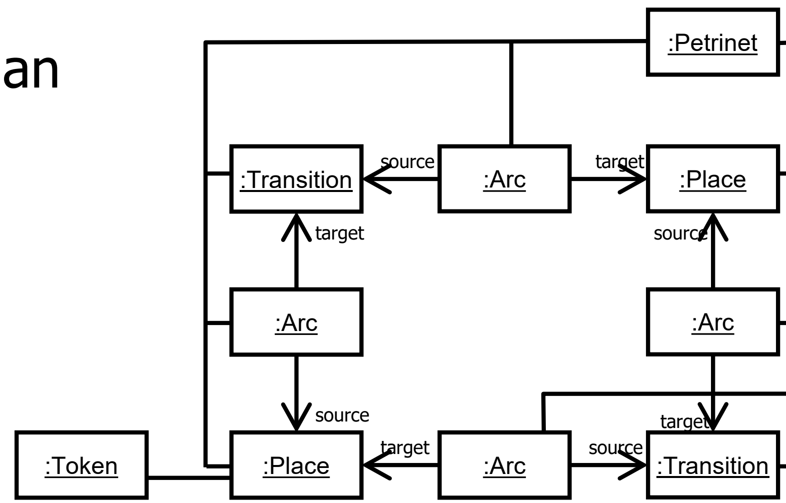




Eclipse EMF and GMF technology

generate an editor



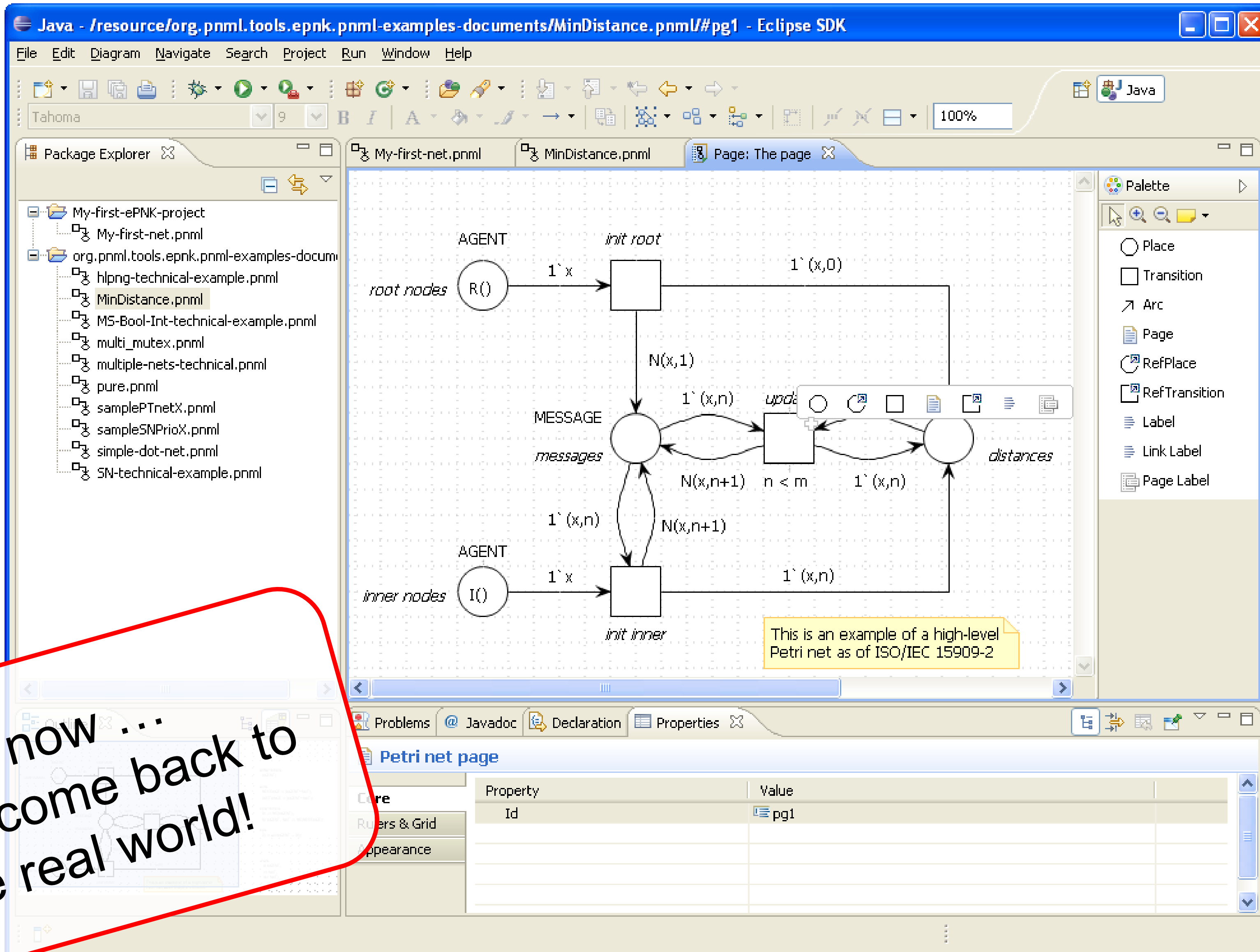


concrete syntax

abstract syntax

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



And now … welcome back to the real world!

Real world issues

- Many more features:
  Pages, reference nodes, ...
- Need to define specific XML syntax (PNML)
- Different versions of Petri nets
  (each would need a separate GMF-editor)
- Definition of new versions of Petri net types
  (without touching the existing tool,
  without programming at all?)

as well as some other extensions

Outline

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Example: YAWL nets and simulator

PNML in a nutshell

- The Petri Net Markup Language (PNML) is an XML-based transfer format for "all kinds" of Petri nets.

- PNML is an International Standard: ISO/IEC-15909-2
  Part 2: focus on high-level nets (under ballot – again)
  Part 3: different extensions
  - modularity
  - type and feature definitions
  - particular versions of Petri nets
  - …

Note that Part 3 is not an international standard yet.

But ...

Isn't XML just soooo boring?

You are sooo right!

That's why the focus is on concepts.

PNML and XML

- The Petri Net Markup Language (PNML) is an XML-based transfer format for "all kinds" of Petri nets.
- For exchanging, PNML between different tools, the XML syntax is important; but that's a technical issue.
- **The interesting stuff are the concepts of PNML.**

Challenges

many versions and variants of Petri nets

- with many common features,
- but also with many variations,
- some fundamental differences,
- and many different combinations of the same or similar features

Petri nets are so simple that everybody thinks he or she can easily change them.

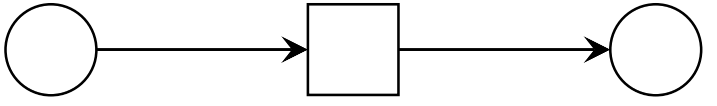
Objective

- PNML should enable the exchange of all kinds of Petri nets, and, ultimately,
- alleviate exchanging between Petri net tools that support different versions of Petri nets without loosing too much information.

A first example

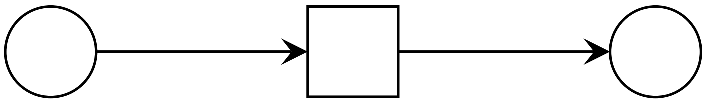
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
<place id="p1"/>
<arc id="a1" source="p1" target="t1"/>
<transition id="t1"/>
<arc id="a2" source="t1" target="p2"/>
<place id="p2"/>
```



A first example

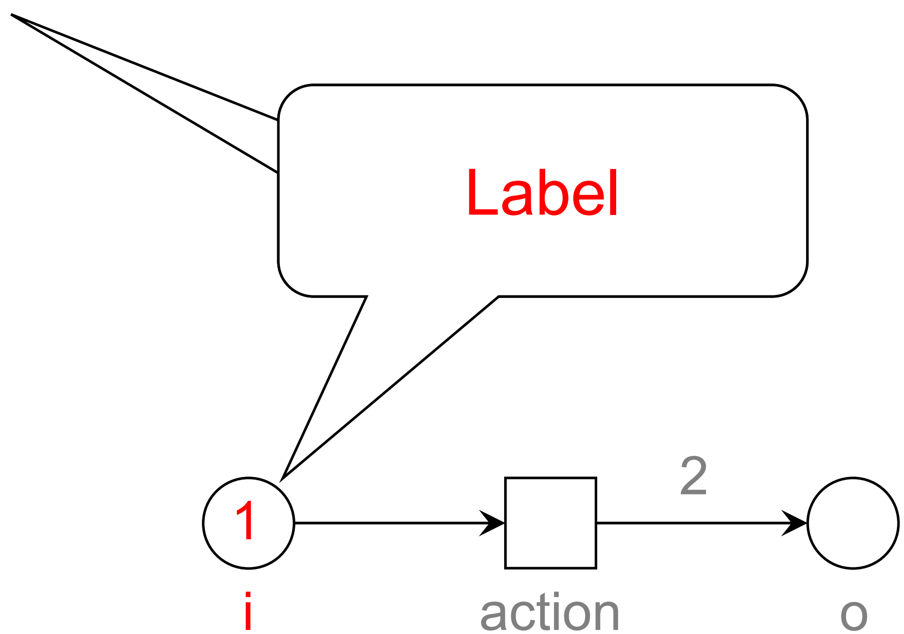

```
<pnml xmlns="http://www.pnml.org/...">
  <net id="n1" type="...">
    ...
    <place id="p1"/>
    <arc id="a1" source="p1" target="t1"/>
    <transition id="t1"/>
    <arc id="a2" source="t1" target="p2"/>
    <place id="p2"/>
    ...
  </net>
</pnml>
```



A first example


```
...
    <place id="p1">
      <name>
        <text>i</text>
      </name>
      <initialMarking>
        <text>1</text>
      </initialMarking>
    </place>
...
```

The particular kind of label depends on the „kind“ of Petri net.



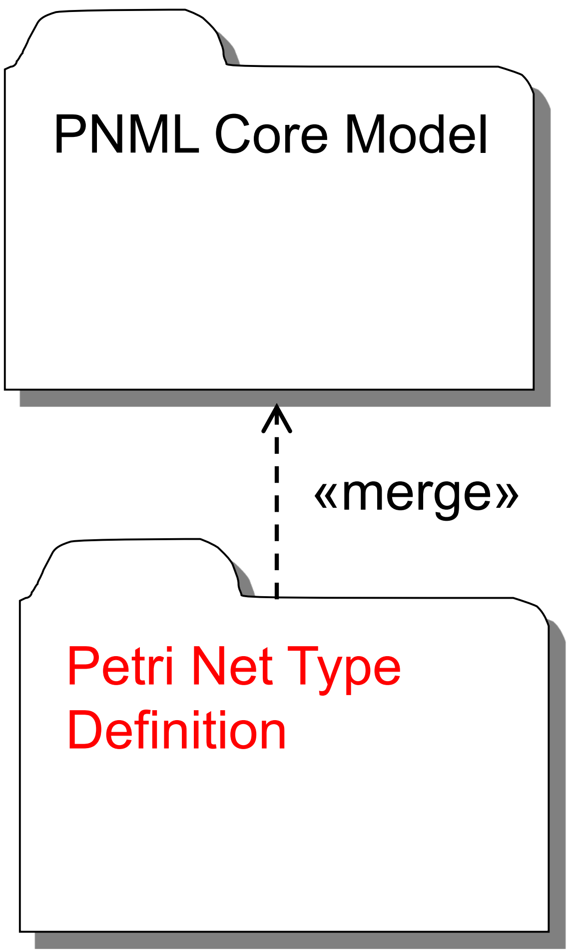
Basic Idea

„All kinds“ of Petri nets can be represented by

- places
- transitions, and
- arcs

along with some

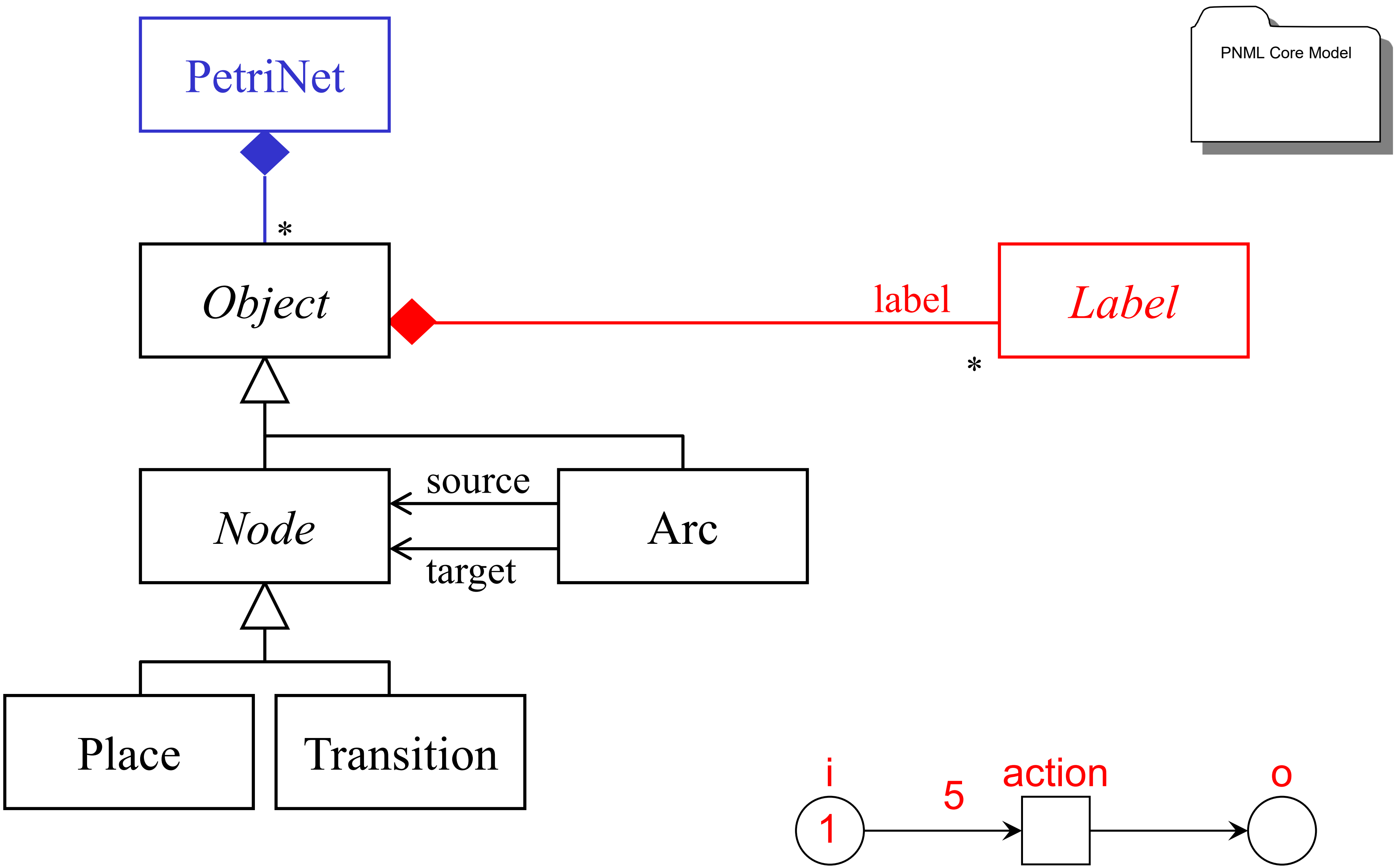
- labels



Outline

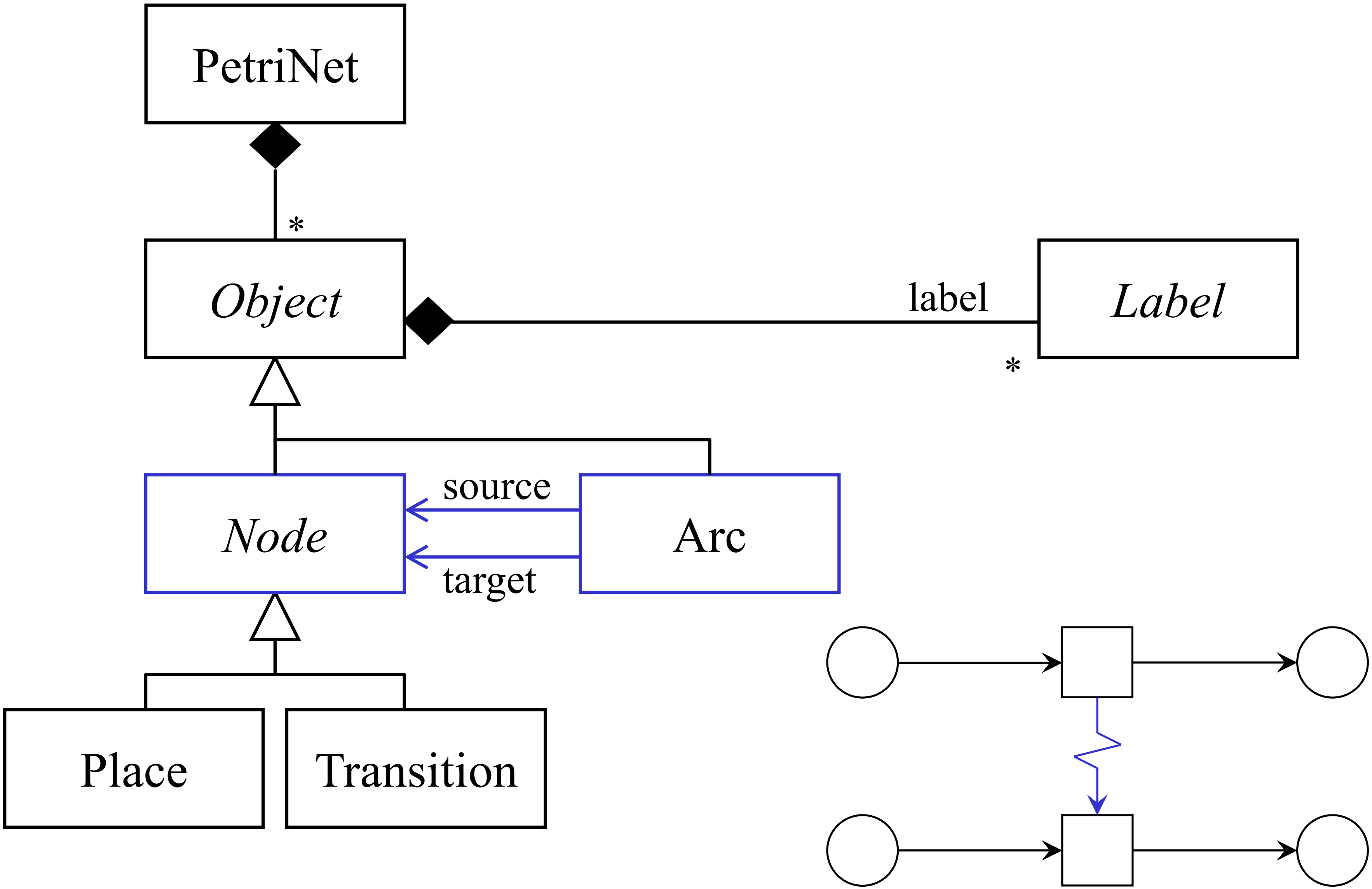
- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Example: YAWL nets and simulator

Core Model (overview)

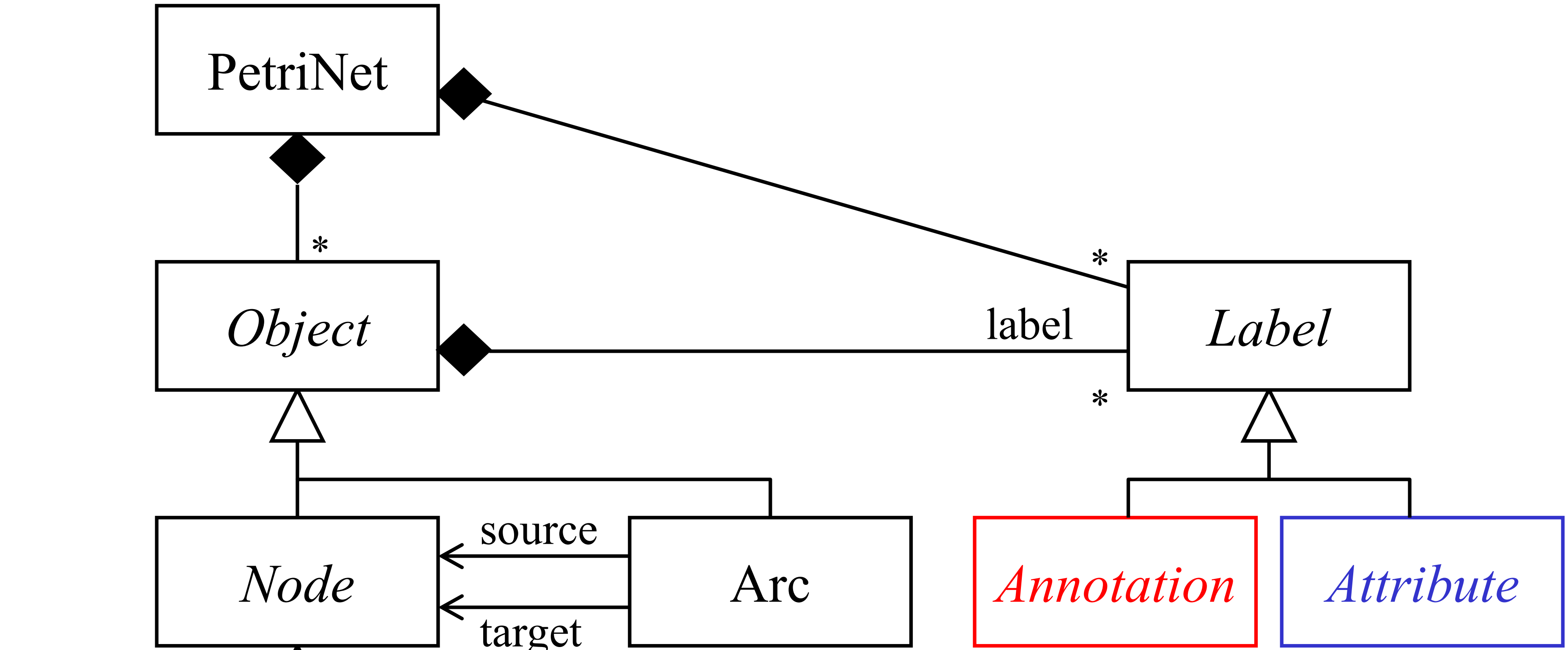


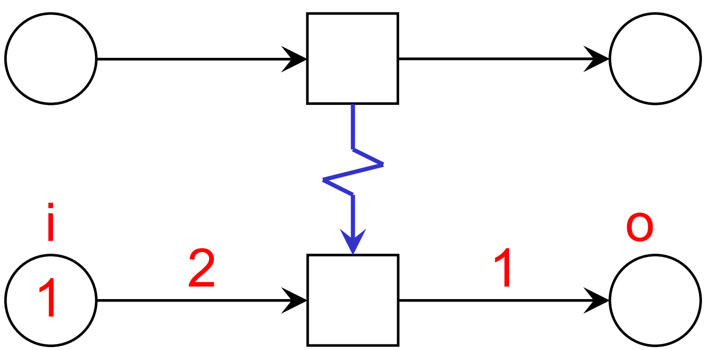
Core Model (overview)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



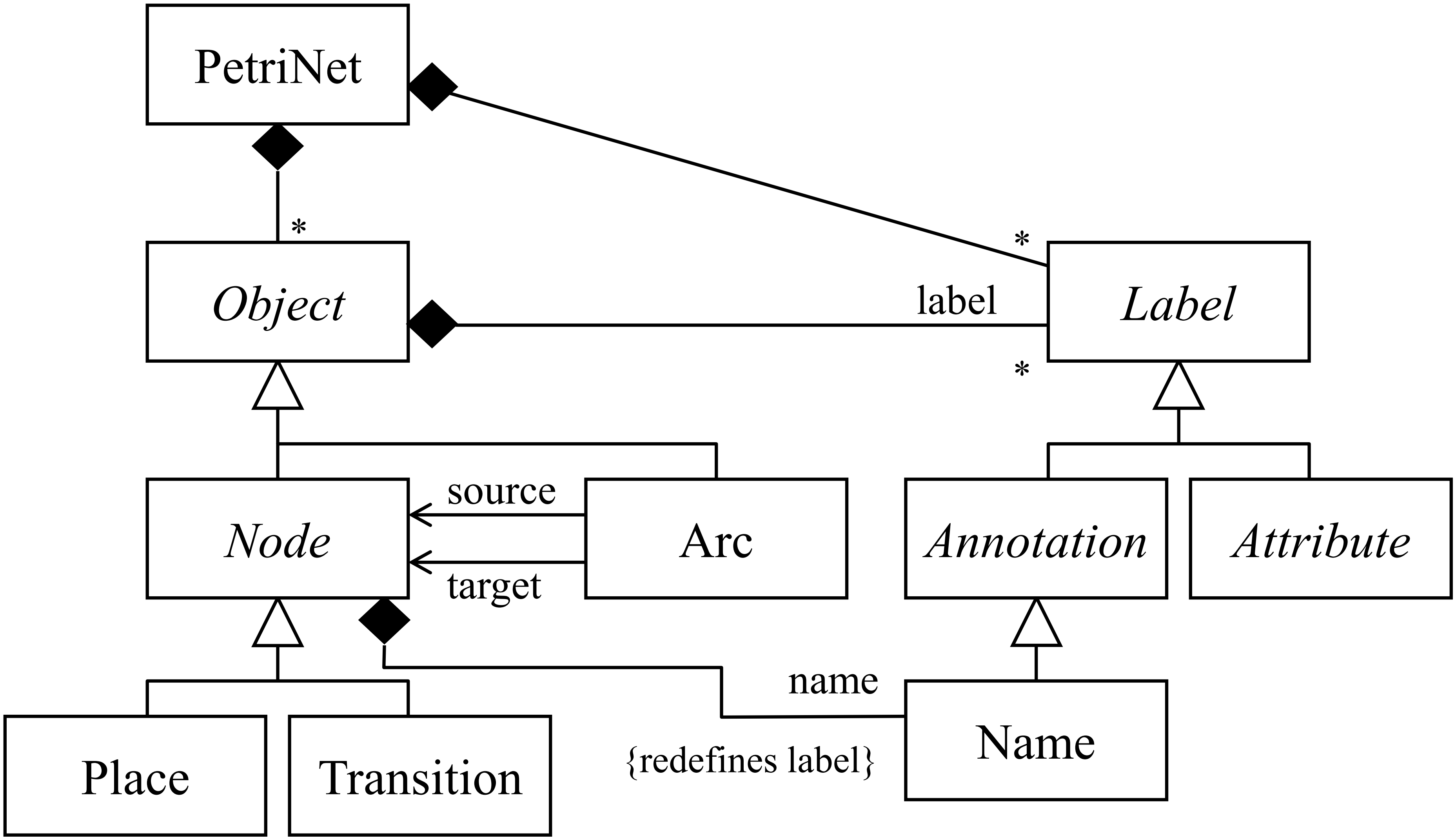
Core Model (overview)



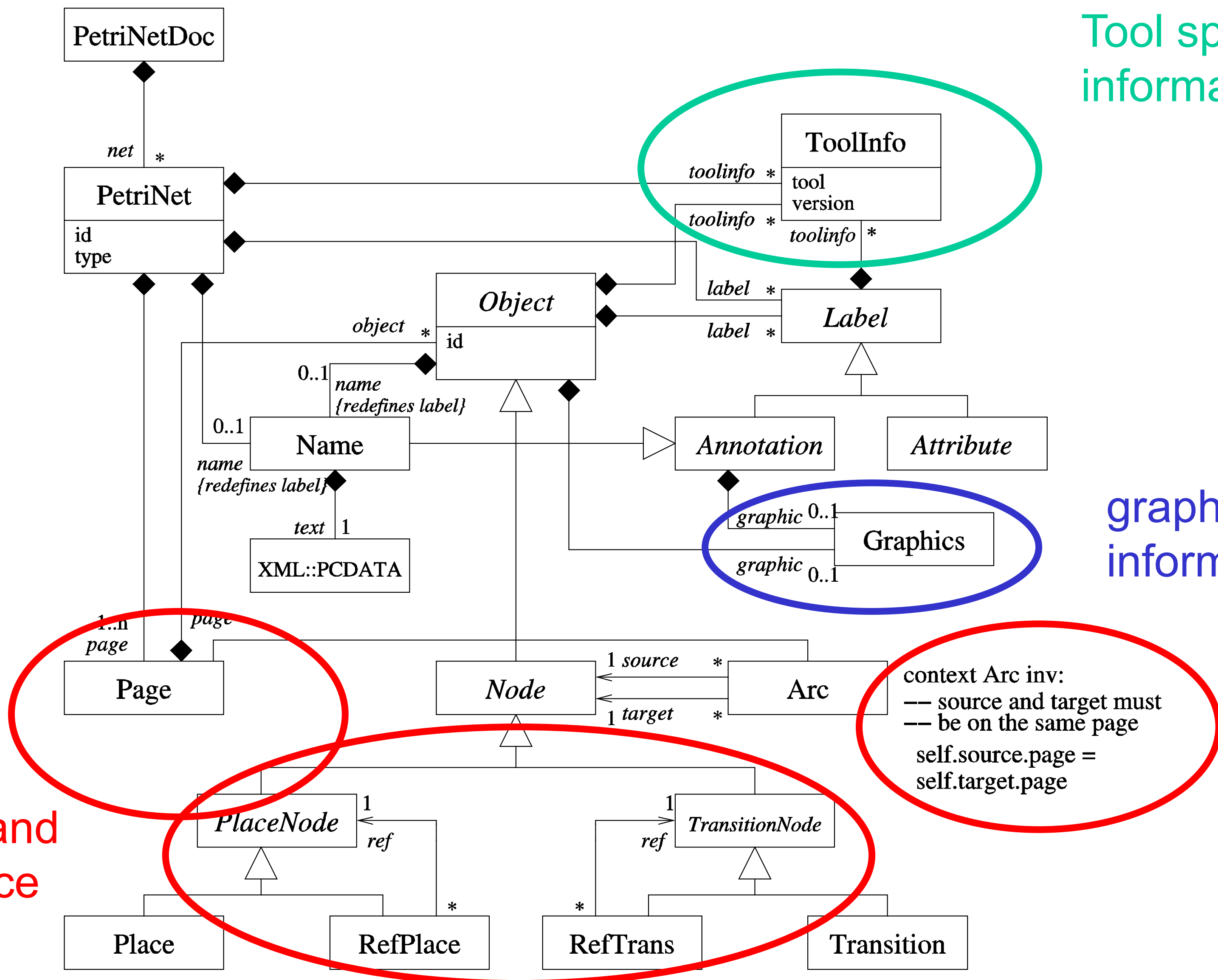


Core Model (overview)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



PNML Core Model



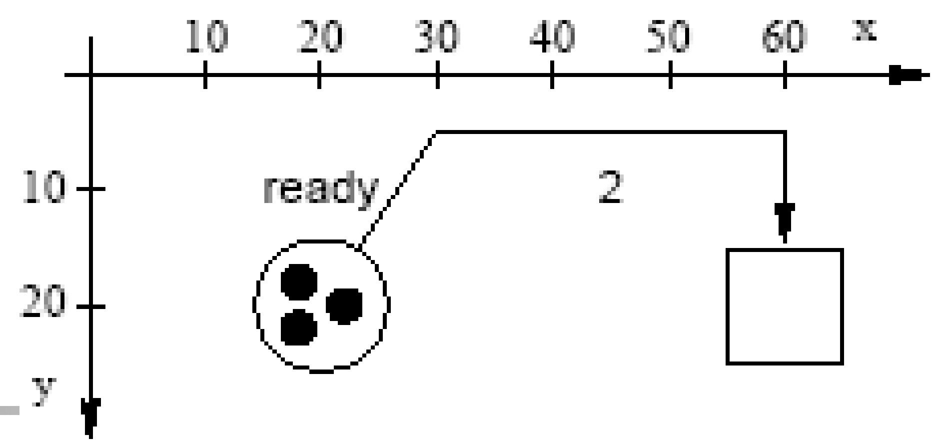
Tool specific information

graphical information

pages and reference nodes

Tool specific information

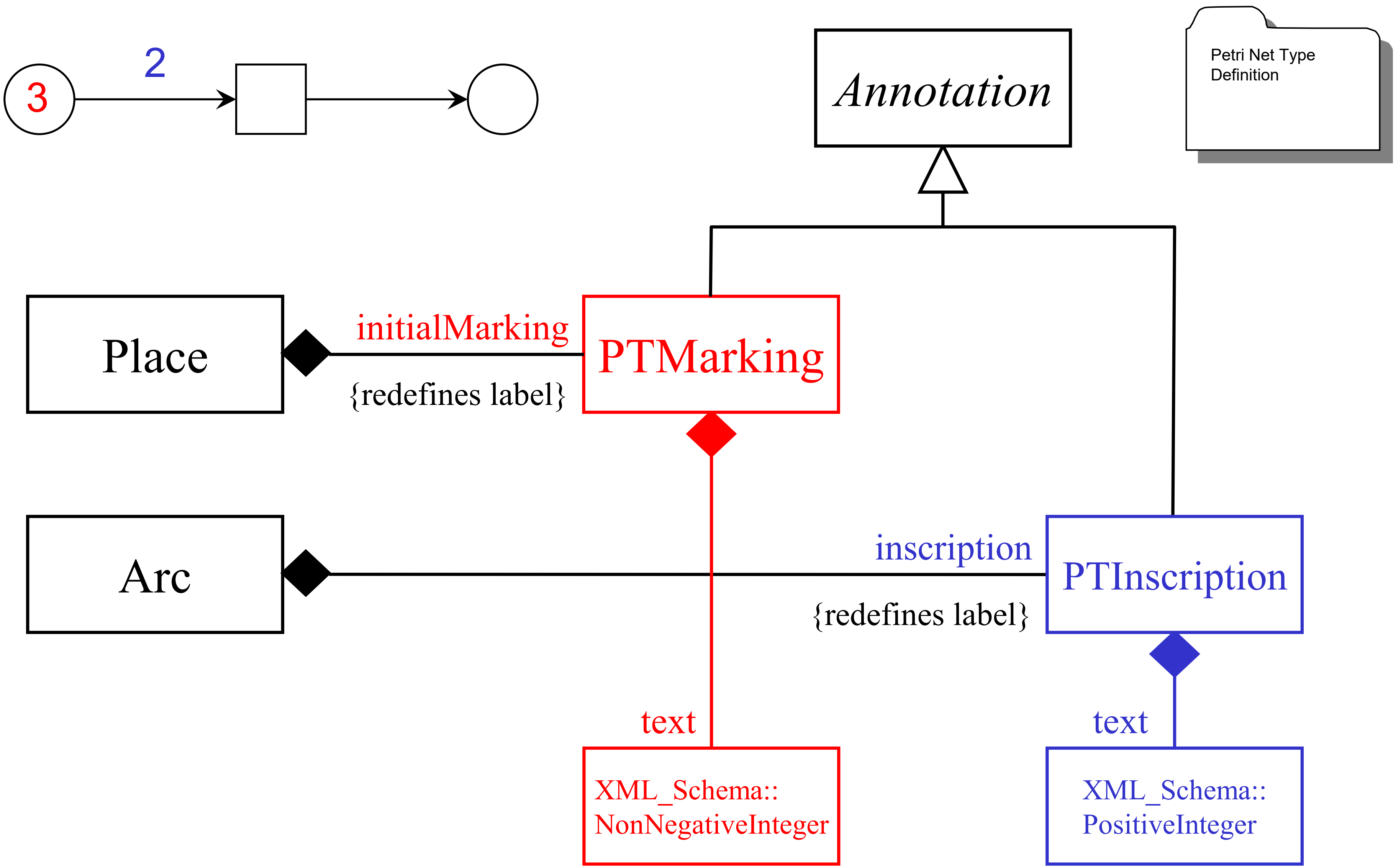
```
<initialMarking>
  <text>3</text>
  <toolspecific tool="org.pnml.tool"
                version="1.0">
    <tokengraphics>
      <tokenposition x="-2" y="-2" />
      <tokenposition x="2" y="0" />
      <tokenposition x="-2" y="2" />
    </tokengraphics>
  </toolspecific>
</initialMarking>
```



Outline

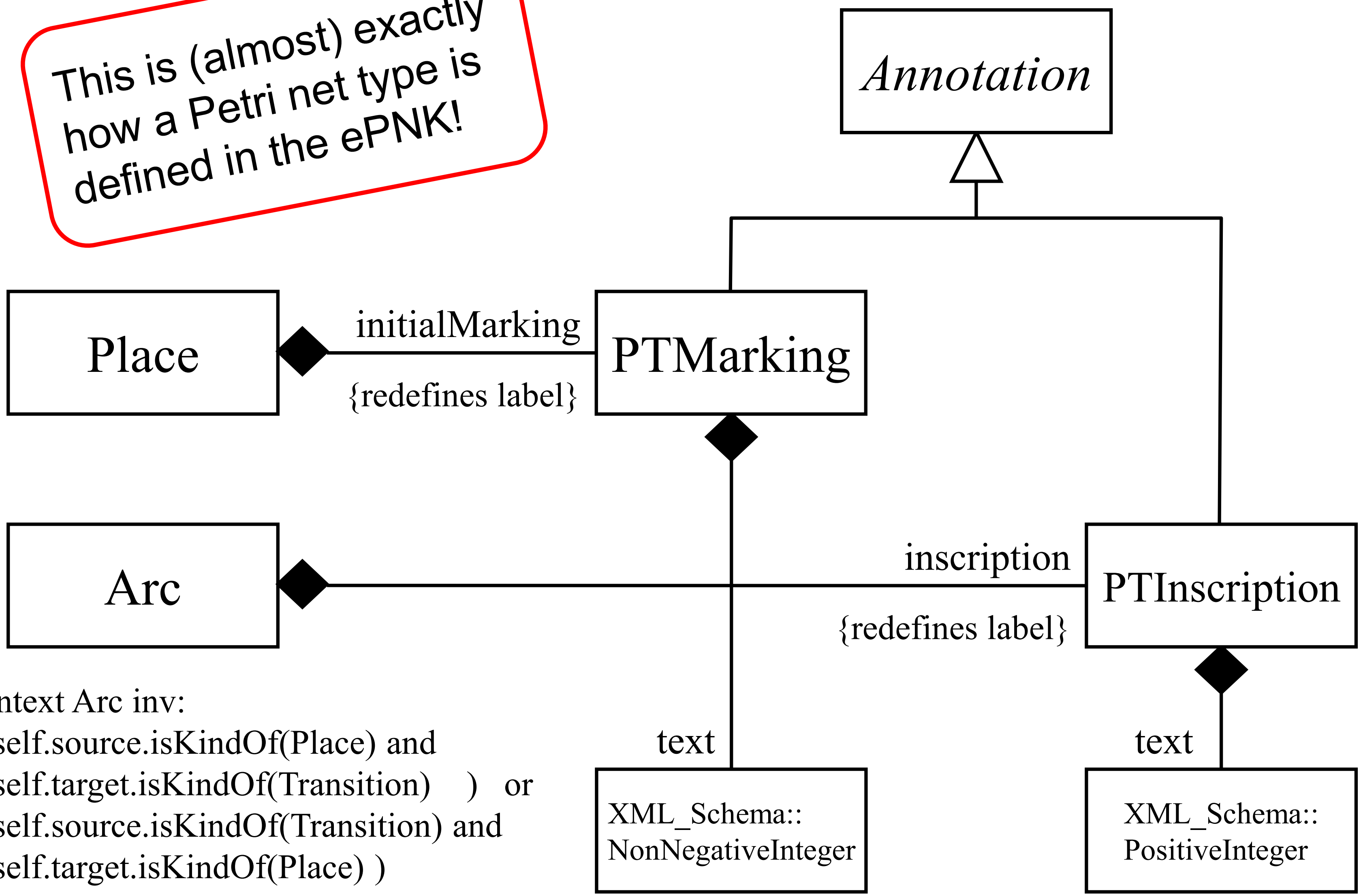
- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Example: YAWL nets and simulator

Type Definition: PT-Net



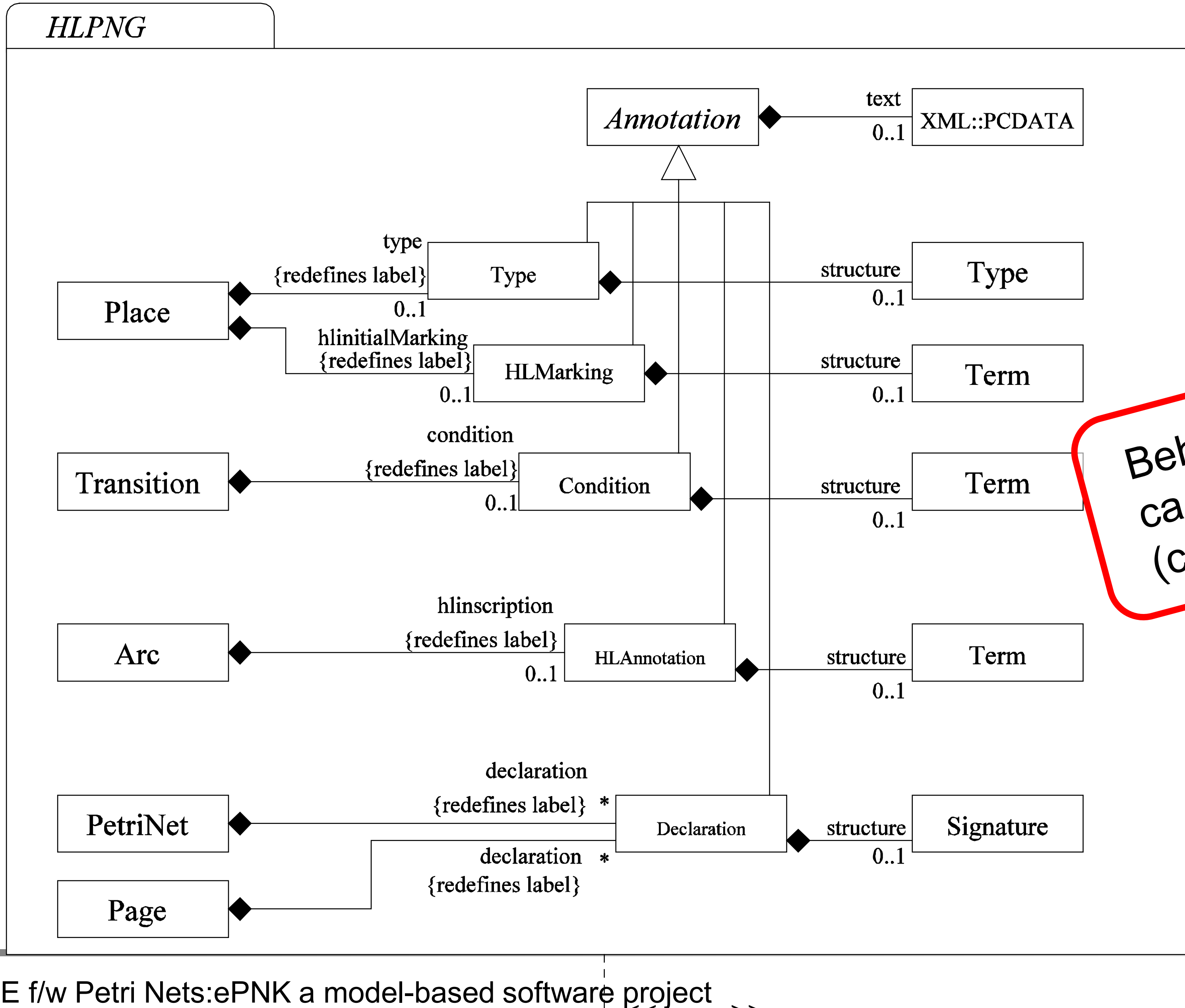
Type Definition: PT-Net

This is (almost) exactly how a Petri net type is defined in the ePNK!



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

Type Definition: HLPNG (overview)



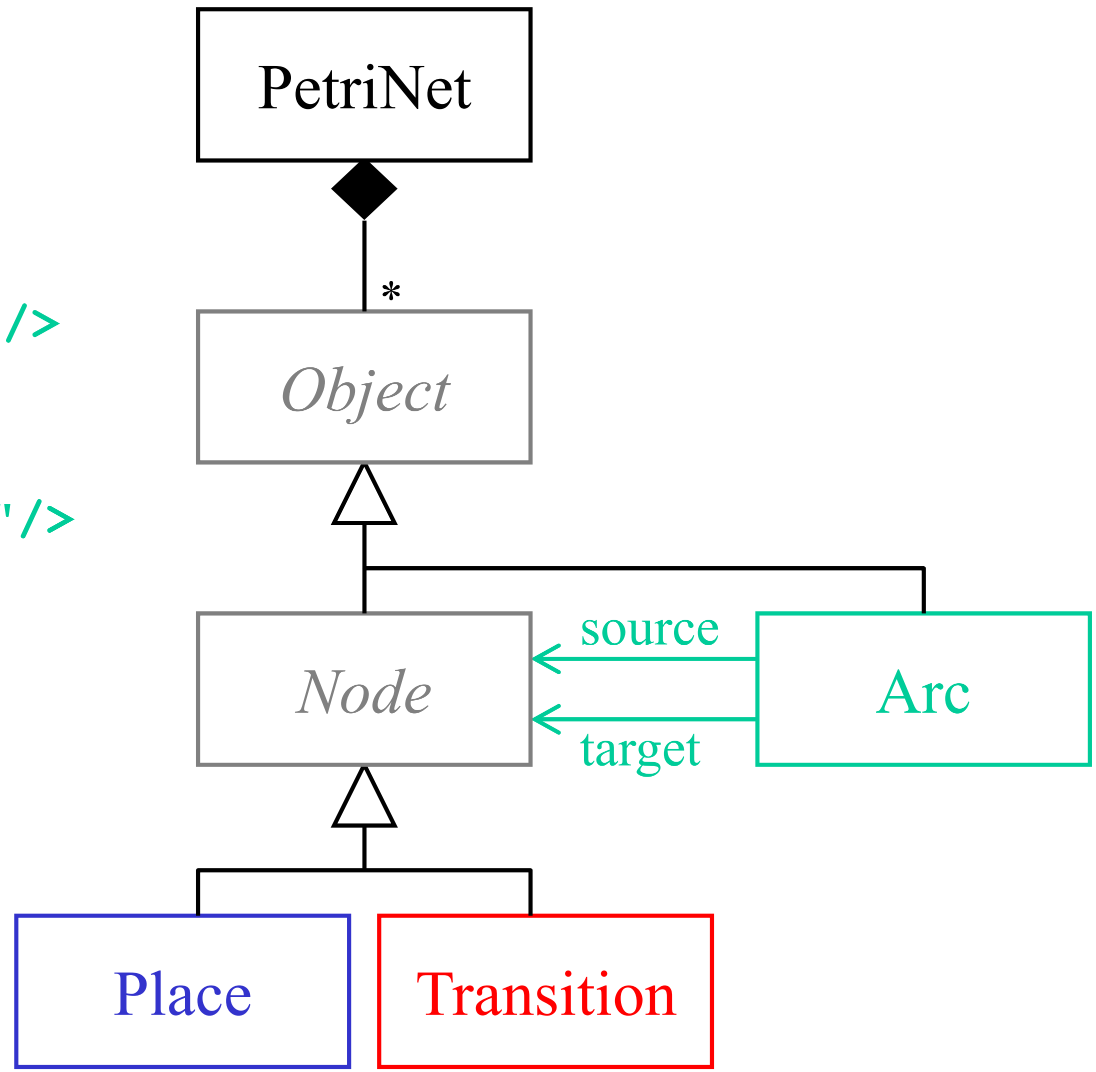
Behind this part ca. 80 classes (constructs).

Outline

- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Experience and statistics

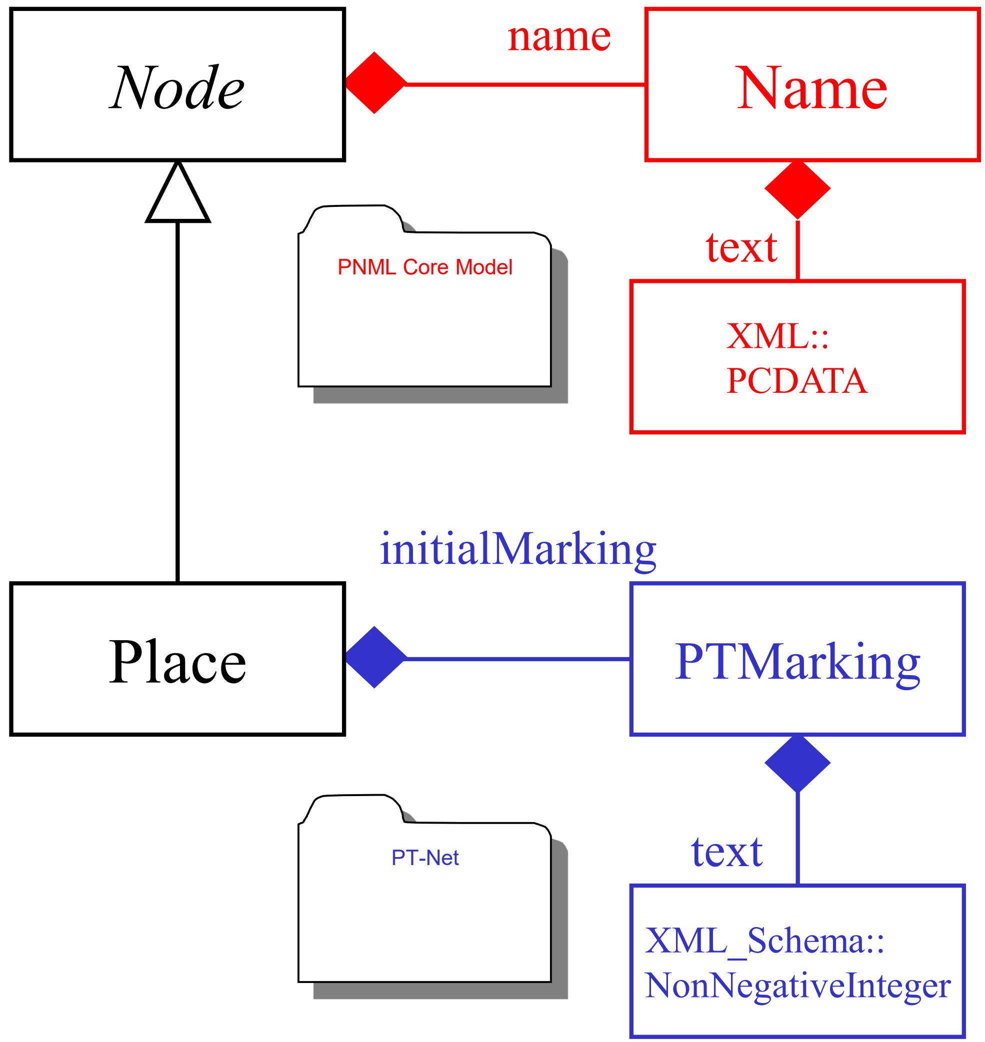
Core Model in XML

```
<pnml xmlns="http:...">
 <net id="n1" type="...">
  <place id="p1"/>
  <arc id="a1" source="p1"
               target="t1"/>
  <transition id="t1"/>
  <arc id="a2" source="t1"
                target="p2"/>
  <place id="p2"/>
 </net>
</pnml>
```



Labels in XML


```
...
  <place id="p1">
    <name>
      <text>i</text>
    </name>
    <initialMarking>
      <text>1</text>
    </initialMarking>
  </place>
...
```



In general

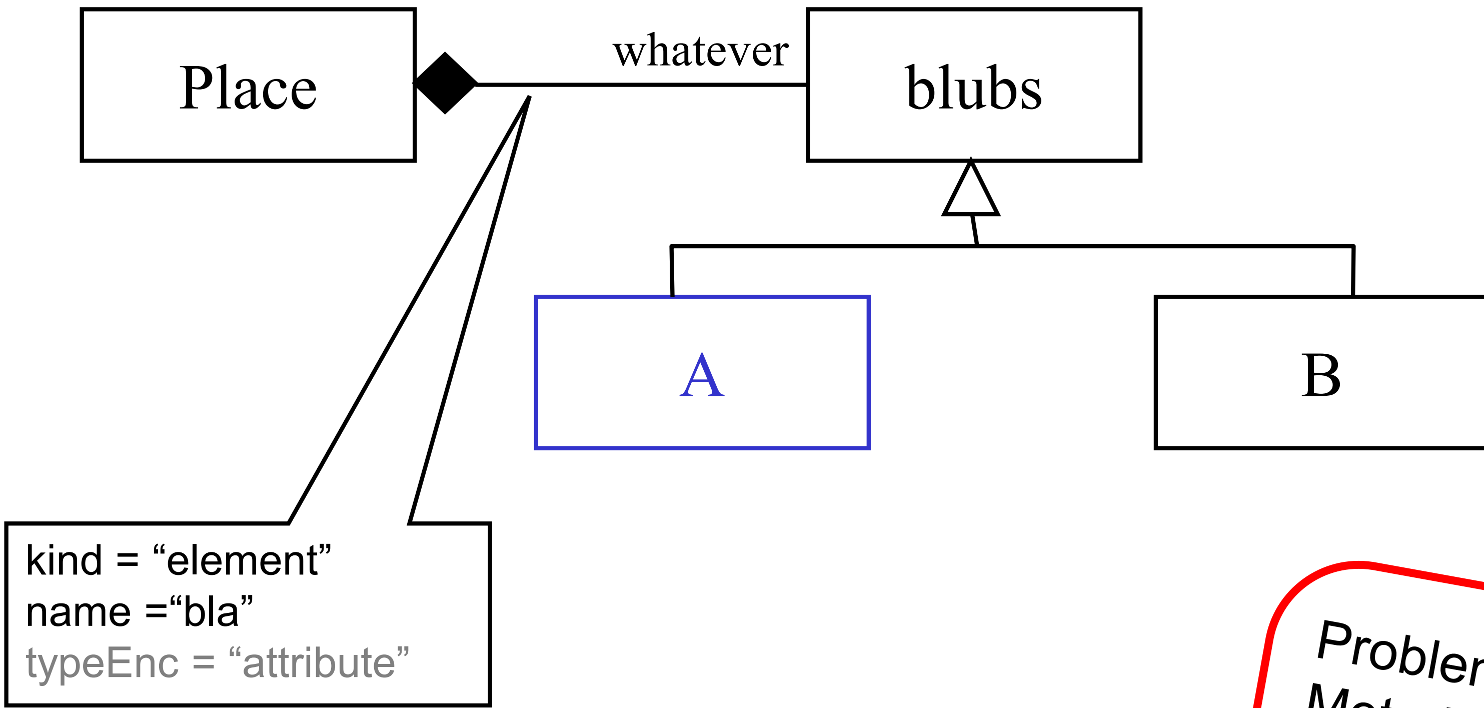
- How can this be defined in general?
  - Core model:
    Just implement it
  - Petri net type:
    Just implement it
    - code it for every new type!
    - interface with rest?

Better idea: use infrastructure to map model concepts to XML (ExtendedMetadata)!?

Outline

- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- **Problems and issues**
- Concepts for solutions
- Example: YAWL nets and simulator

Idea: Extended Metadata




```
<place ...>
  <bla type="...ptnets.A">
    <text>... </text>
  </bla>
</place>
```

Problem: Extended Metadata is not powerful enough to express all mappings necessary for PNML. ePNK implements its own mapping mechanism.

Problems

- Mapping from concepts (model) to XML (and vice versa)

- How to plug in net type models and their XML mapping
- Implement a complex type (>= 80 classes) and a complex concrete syntax in a simple way – and complex conditions

- How to plug in tool-specific features and use standard XMI mapping along with PNML-serialization
- How to deal with unknown tool-specific extensions (ignore them without deleting them)

- One graphical editor for all (also future) Petri net types (generic graphical editor)

→ using Model-based Software Engineering technologies (reusing as much as possible from EMF, GMF, Xtext, Validation)

Outline

- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Example: YAWL nets and simulator

Solutions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- PNML-Mapping:
  - extended mapping that is flexible enough for all our needs (and some more)
  - hooked into the existing XMI-serialization (→ when there is no mapping defined in the ePNK, XMI is used as default, XMI deals with cross-references even when no id's exist)

- Net-types plug-in:
  - A EMF-model plus a factory for producing all the extended elements
  - PNML-Mapping for new elements (if necessary)
  - Separate constraints for syntactical constraints (batch and live)

  - For structured net types: Interface for parsing and linking (in concrete type used Xtext → worked surprisingly smooth for parsing and surprisingly bad for serialization)

Solutions

- Tool-specific extensions:
  - Plug-in for tool-specific extensions
  - “magic” AnyType hooked into XMI-Mapping/PNML-Mapping (keeps XML structure which you do not care for and writes it again)

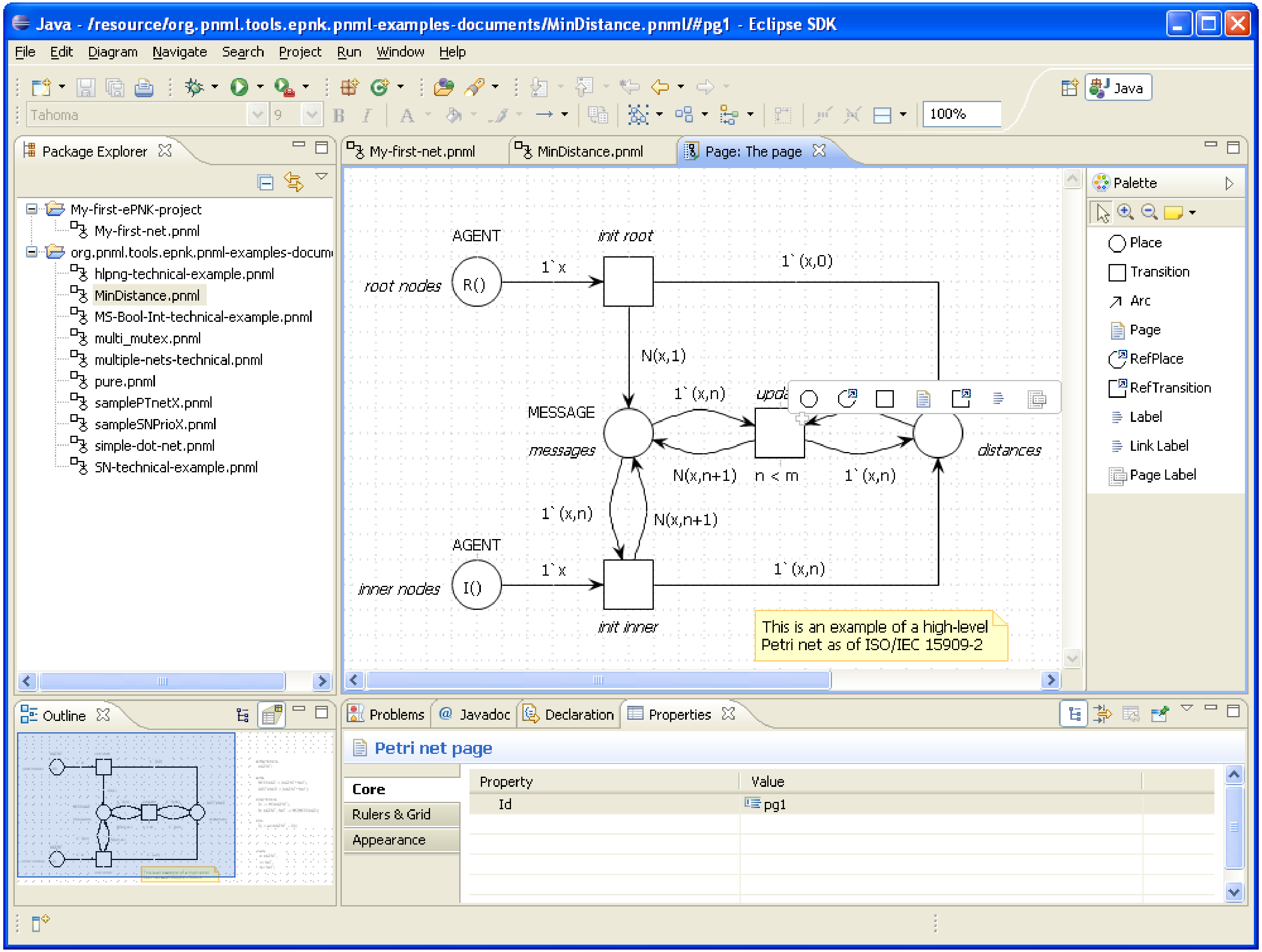
- Graphical editor:
  - Integrated EMF/GMF-Editor (worked surprisingly simple; but many nasty little but time-consuming issues)
  - ProxyLabels that, below the surface, can be any Petri Net Type specific label (using a reflective API to get the right ones)

  - Explicit generation of GMF diagram from PNML graphical information
  - Update of PNML graphical information via listeners to GMF-diagram

→ all this required many manual changes in the GMF-generated editor

Outcome

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



ePNK: Features

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Generic graphical editor for all kinds of Petri nets
- Supporting PNML, PT-Nets, HLPNGs, SN
(some graphical information still ignored)
- Problem reporting mechanism
- Some basic functionality (mostly for demo purposes)
  - simple simulator for PT-Nets
  - simple codegenerator for PT-Nets
  - simple model checker for PT-Nets
  - serialiser for HLPNG labels
(in case PNML nets come without textual labels)

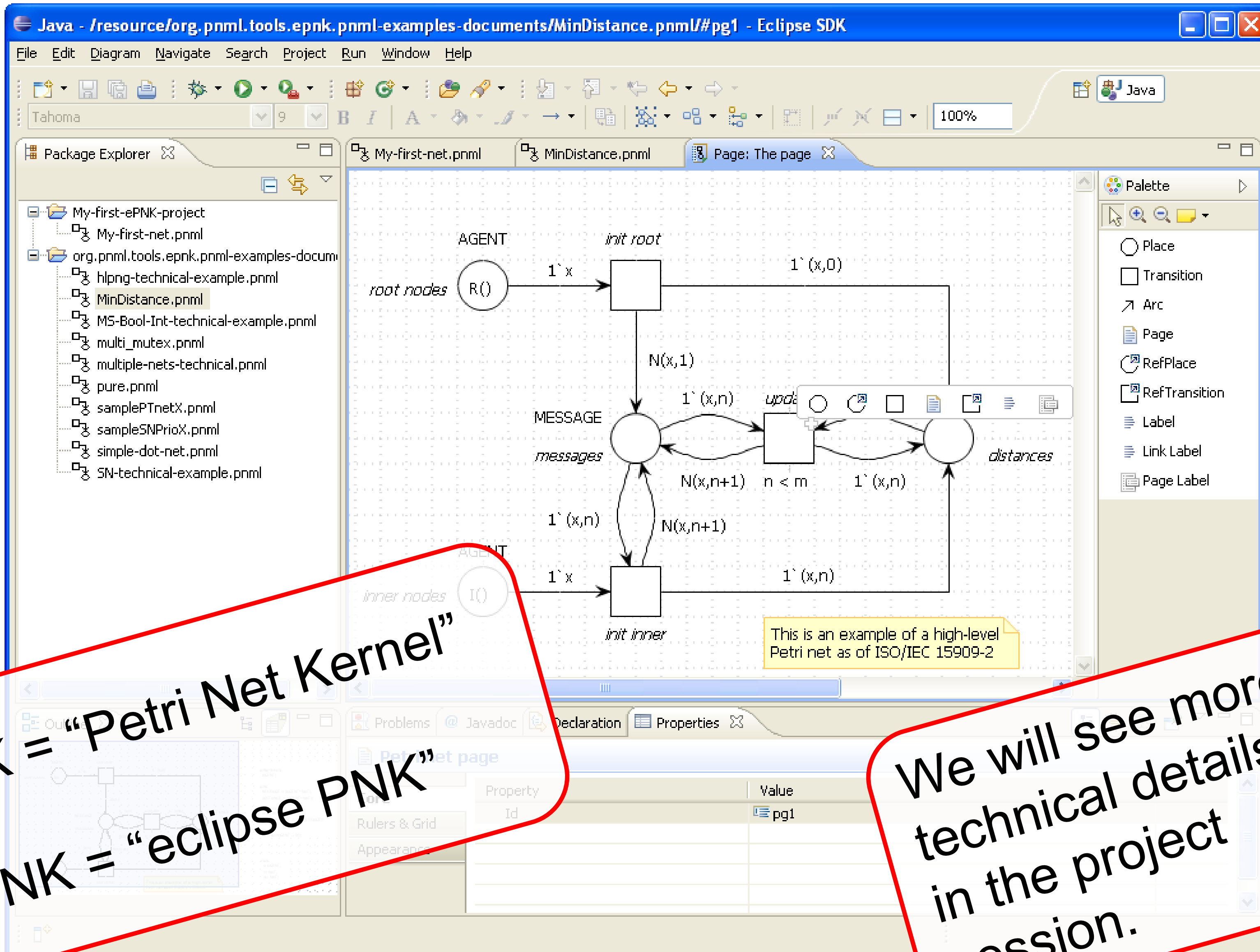
ePNK: Features

- Extension mechanisms
  - defining new net types (basically, making a model) (with or without dedicated mapping to XML for new concepts)
  - constraints for net types (OCL or programmed constraints)
  - graphical appearance of nets and their elements (depending on attributes: inhibitor arcs, read arcs, tokens)
  - tool specific information (basically, making a model)

  - adding new functions (mostly the eclipse plugin mechanism)
  - define ePNK applications, with user interactions

The ePNK

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



PNK = "Petri Net Kernel"
ePNK = "eclipse PNK"

We will see more technical details in the project session.

Outline

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

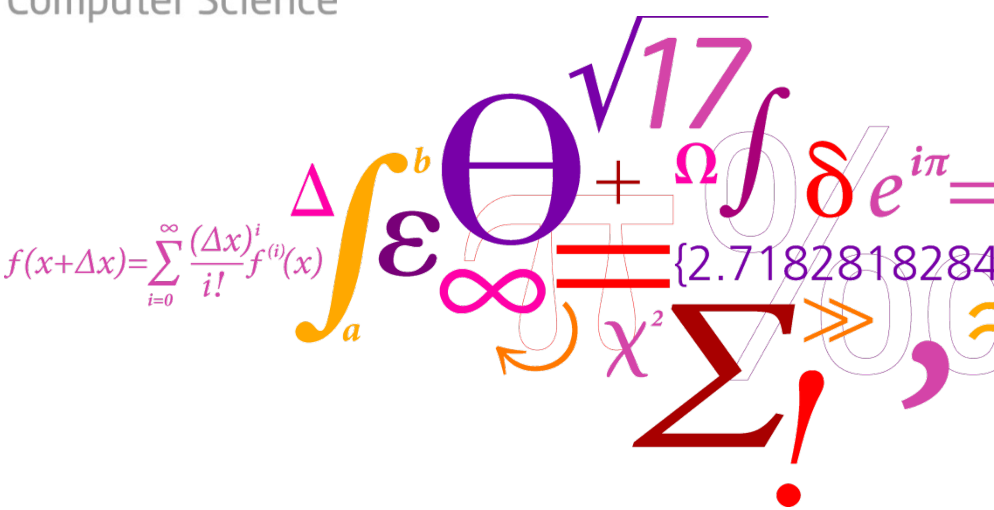
- Motivation
- PNML
  - Overview
  - Core model
  - Type model
  - Mapping to XML
- Problems and issues
- Concepts for solutions
- Example: YAWL nets and simulator

The ePNK:
An Example: YAWL nets and Simulator

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

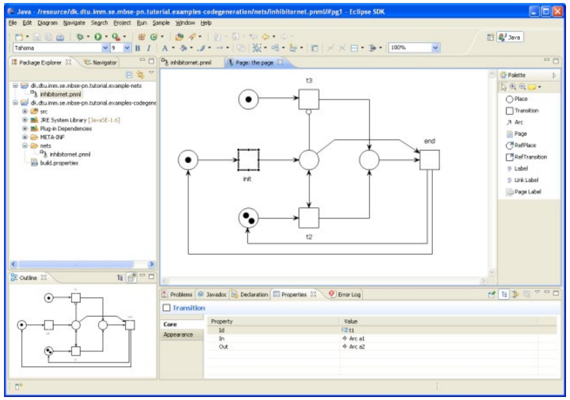
This example is deployed with the ePNK! We can have a look into that in the hands-on part.

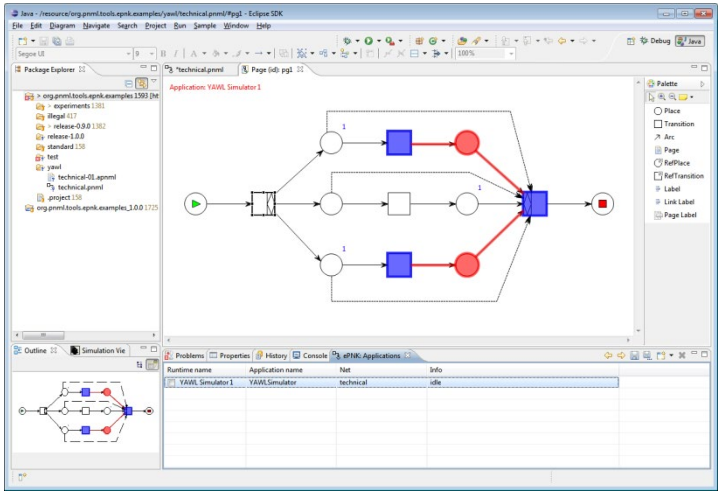


ePNK

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Platform for developing Petri net tools based on the PNML transfer format
- With PNML (core model) at its heart
- Pluggable architecture:
  - any new type of Petri net (PNTD)
  - new of application with visual feedback and user interaction

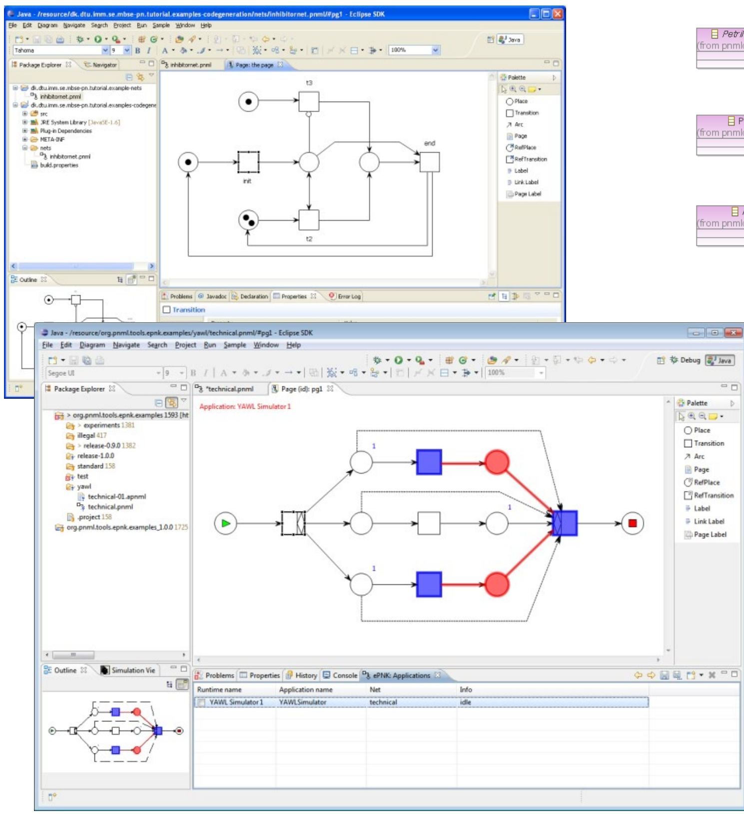


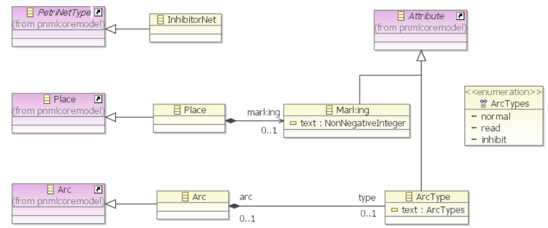


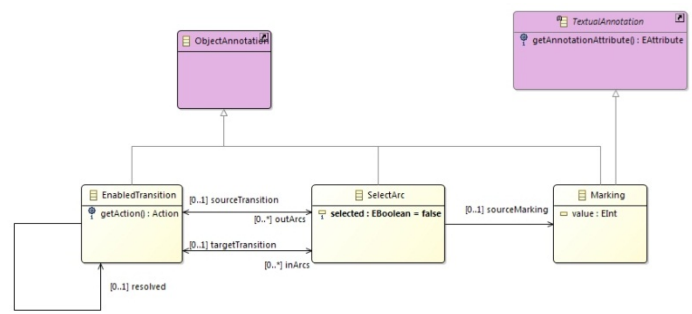
ePNK: Core Paradigm

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Core paradigm: Model-based Software Enigneering

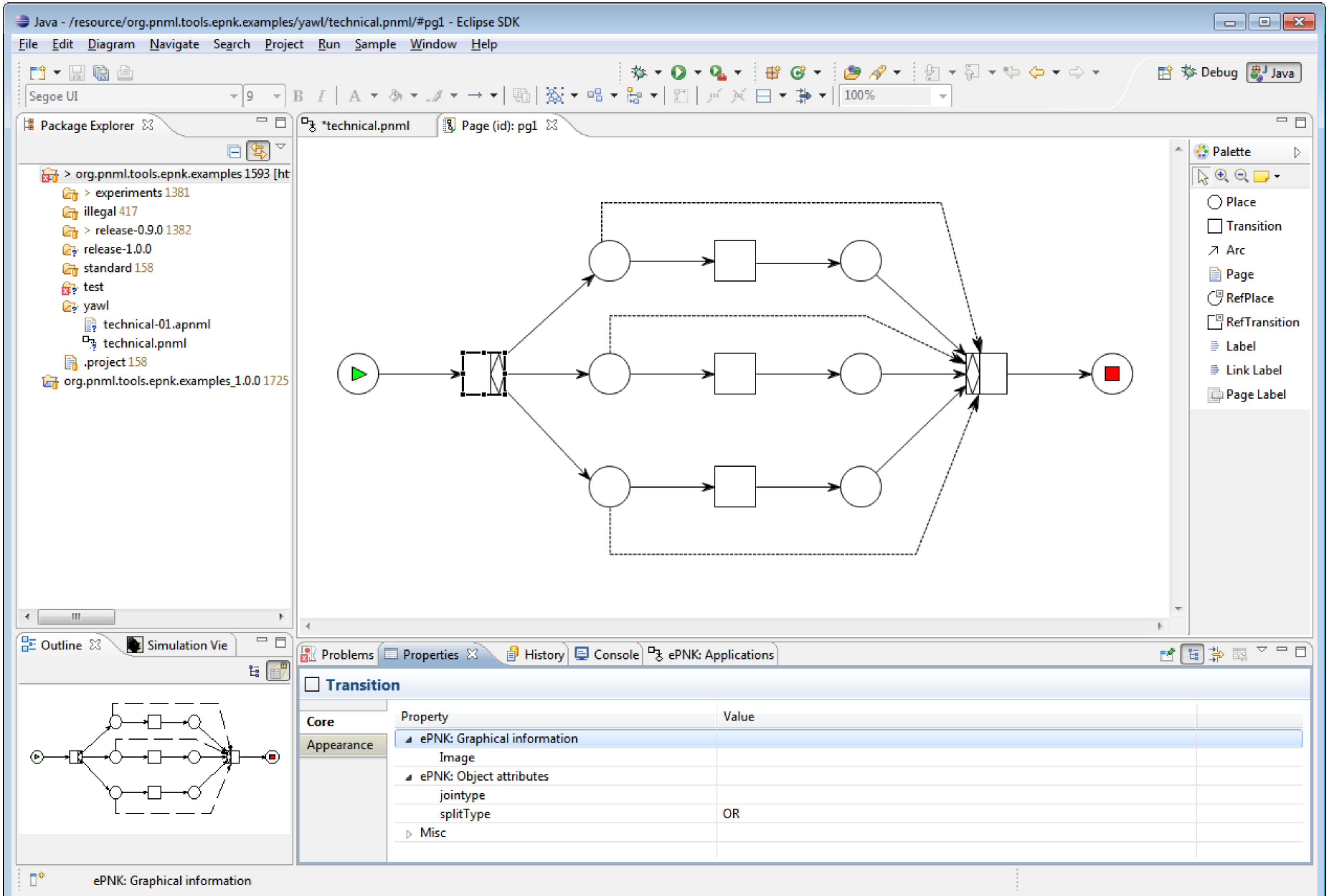






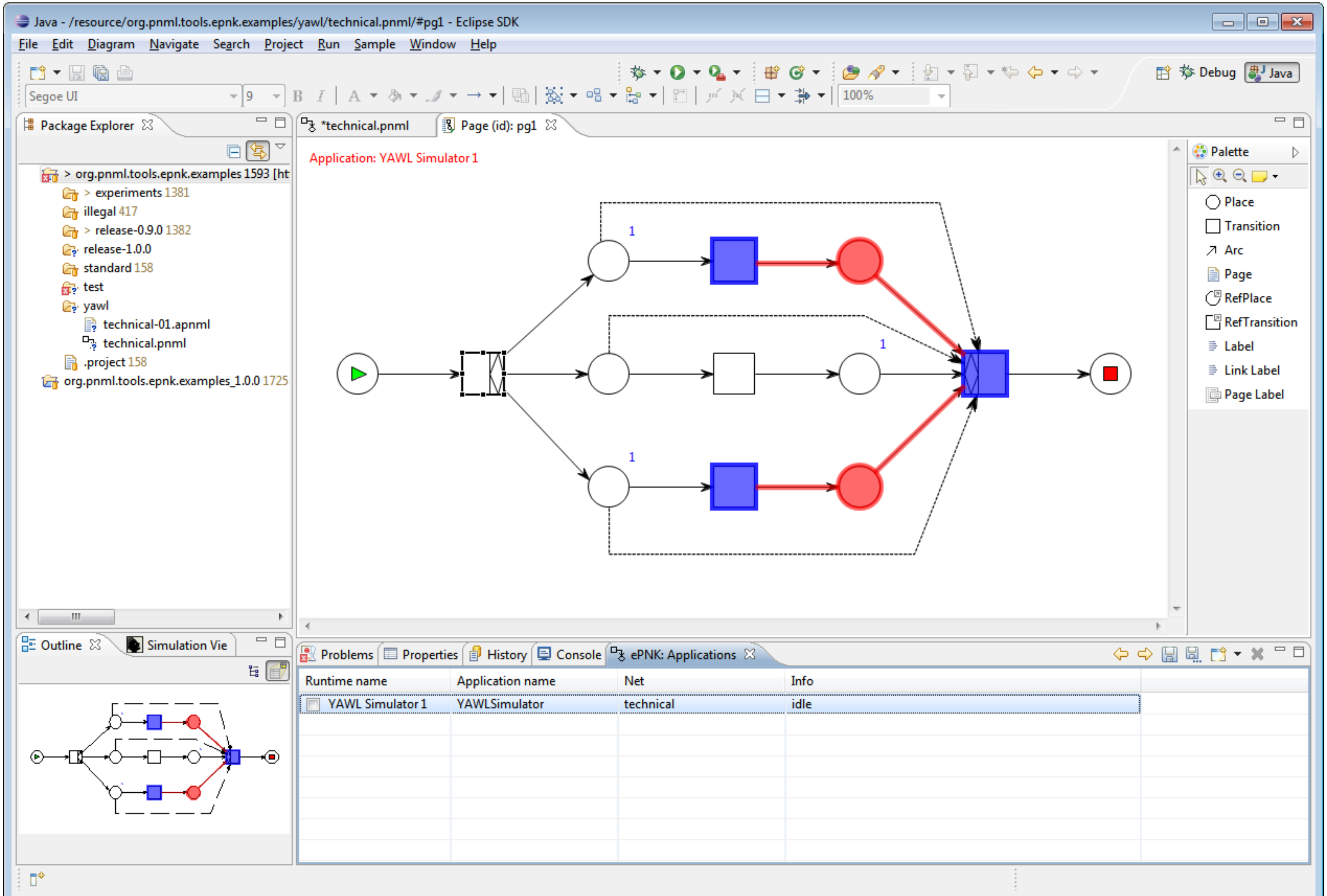
Example: YAWL nets

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Example: YAWL simulator

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



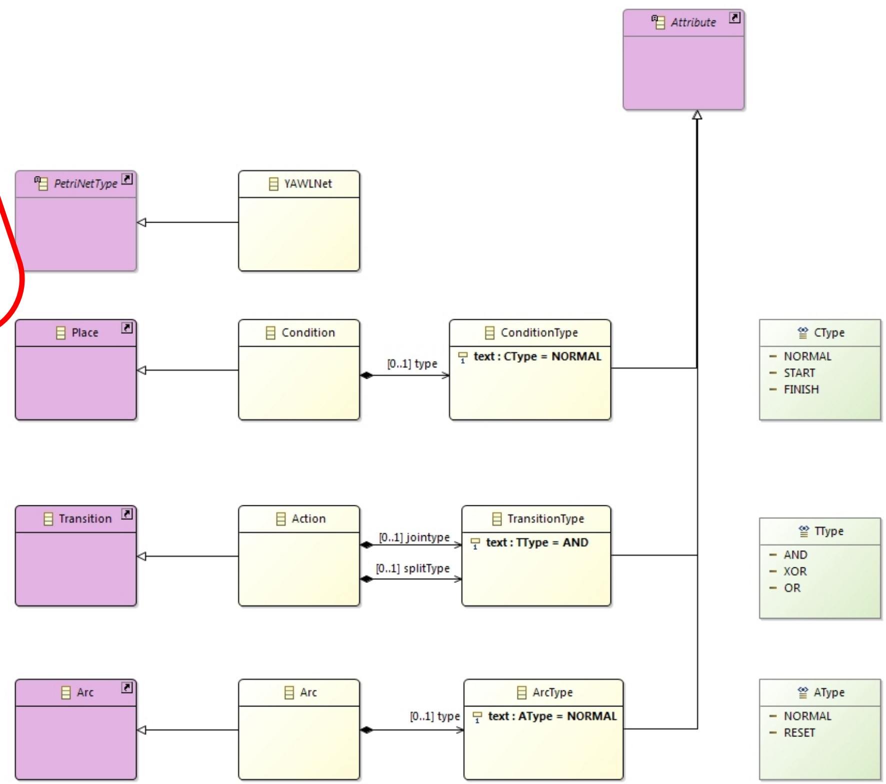
YAWL net: What to do

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Step 1:

- Define the Petri net type by a class diagram (Ecore diagram)

Concepts:
- Conditions with start / finish attribute
- Actions with join and split types
- Arcs with RESET type



YAWL net: What to do

Step 2:

- Define additional constraints

```
( self.source.oclIsKindOf(pnmlcoremodel::PlaceNode) and
  self.target.oclIsKindOf(pnmlcoremodel::TransitionNode) )
  or
( self.source.oclIsKindOf(pnmlcoremodel::TransitionNode) and
  self.target.oclIsKindOf(pnmlcoremodel::PlaceNode) and
  self.type->size() = 0 )
```

Example (here OCL):

- Out-going arcs cannot have the type attribute set (cannot be RESET arcs)

YAWL net: What to do

Example:
- RESET arcs dashed with double arrow head

Step 3:

- Define dedicated graphics

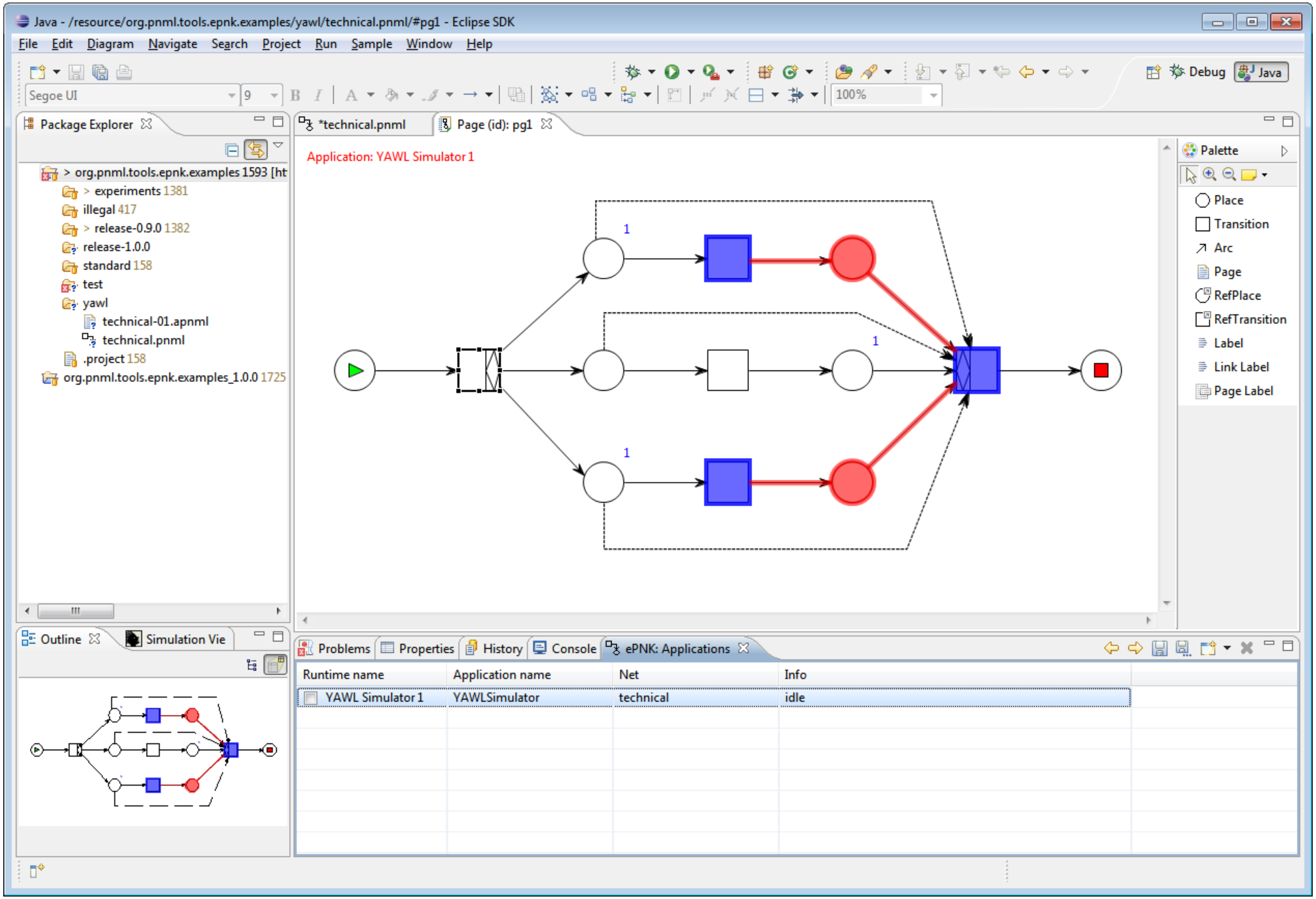
```
public void update() {
    boolean oldIsReadArc = isResetArc;
    isResetArc = YAWLFunctions.isResetArc(arc);
    if (isResetArc != oldIsReadArc) {
        setGraphics();
}   }

private void setGraphics() {
    if (isResetArc) {
        this.setTargetDecoration(
            new DoubleArrowHeadDecoration());
        this.setLineStyle(SWT.LINE_DASH);
    } else {
        this.setTargetDecoration(new ArrowHeadDecoration());
        this.setLineStyle(SWT.LINE_SOLID);
}   }
```

You get

- Graphical editor for YAWL (with dedicated graphic representation of special YAWL features)
- A PNML compatible file format for YAWL along with a save and load operation for that format
- Consistency check for all constraints (live or batch)

Example: YAWL simulator

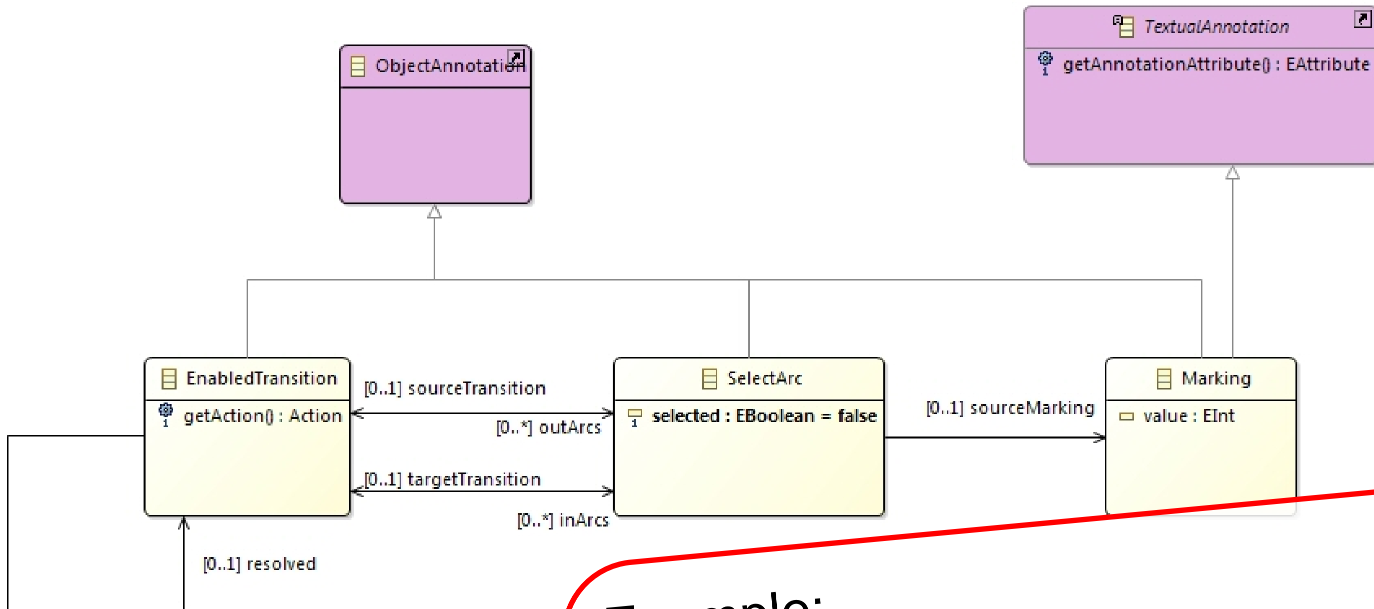
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Simulator: What to do

Step 1:

- Define annotations you need



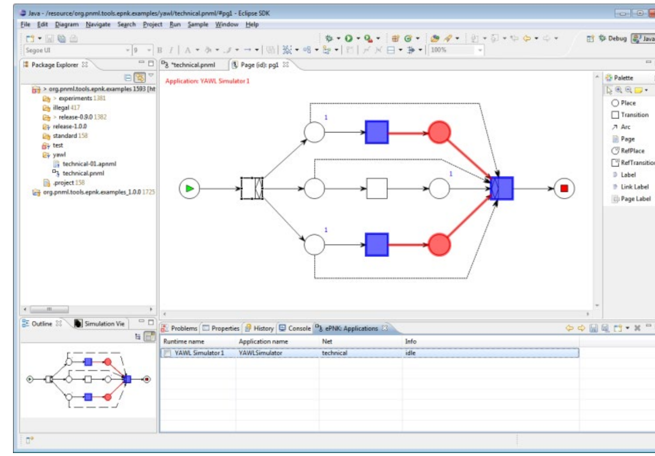
Example:
- Enabled transitions
- Arcs (to be selected)
- Marking of place
- “red marks” for backward cone of OR-joins (standard Object Annotations)

Simulator: What to do

Example:
- Select arcs for XOR-join and for OR- and XOR-splits
- Firing transitions

Step 2:

- Define how annotations should look: Presentation handler(s)



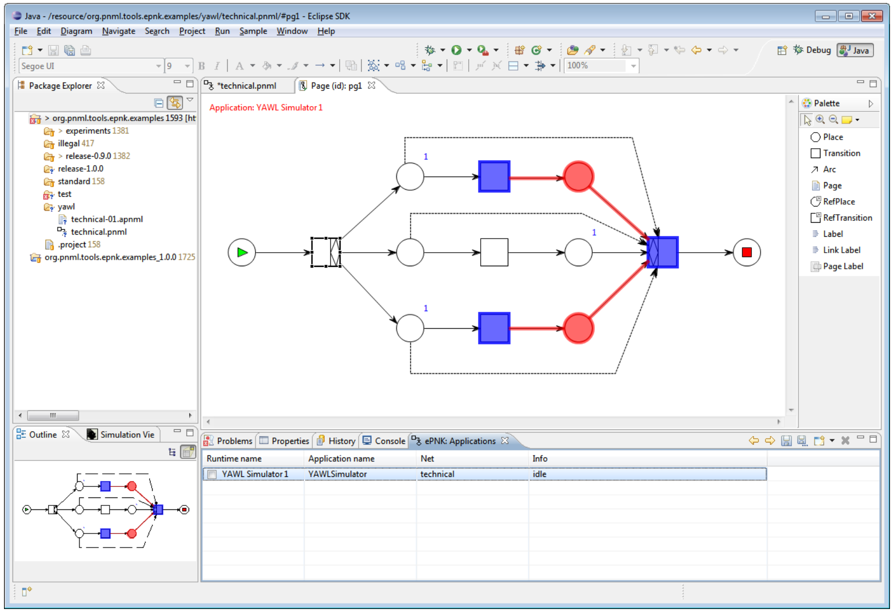
```
if (annotation instanceof SelectArc) {
  SelectArc selectArc = (SelectArc) annotation;
  if (editPart instanceof ConnectionNodeEditPart) {
    ConnectionNodeEditPart connectionEditPart =
      (ConnectionNodeEditPart) editPart;
    Object modelObject =
      connectionEditPart.resolveSemanticElement();
    if (modelObject instanceof Arc) {
    PolylineOverlay overlay = new PolylineOverlay(connectionEditPart);
    if (!selectArc.isSelected()) {
      overlay.setForegroundColor(ColorConstants.lightGray);
      overlay.setBackgroundColor(ColorConstants.lightGray);
    } else {
      overlay.setForegroundColor(ColorConstants.blue);
      overlay.setBackgroundColor(ColorConstants.blue);
   }
   return overlay;
```

Example: YAWL simulator

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Step 3:

- Define what should happen when user clicks / double clicks on an annotation: Action handler(s)

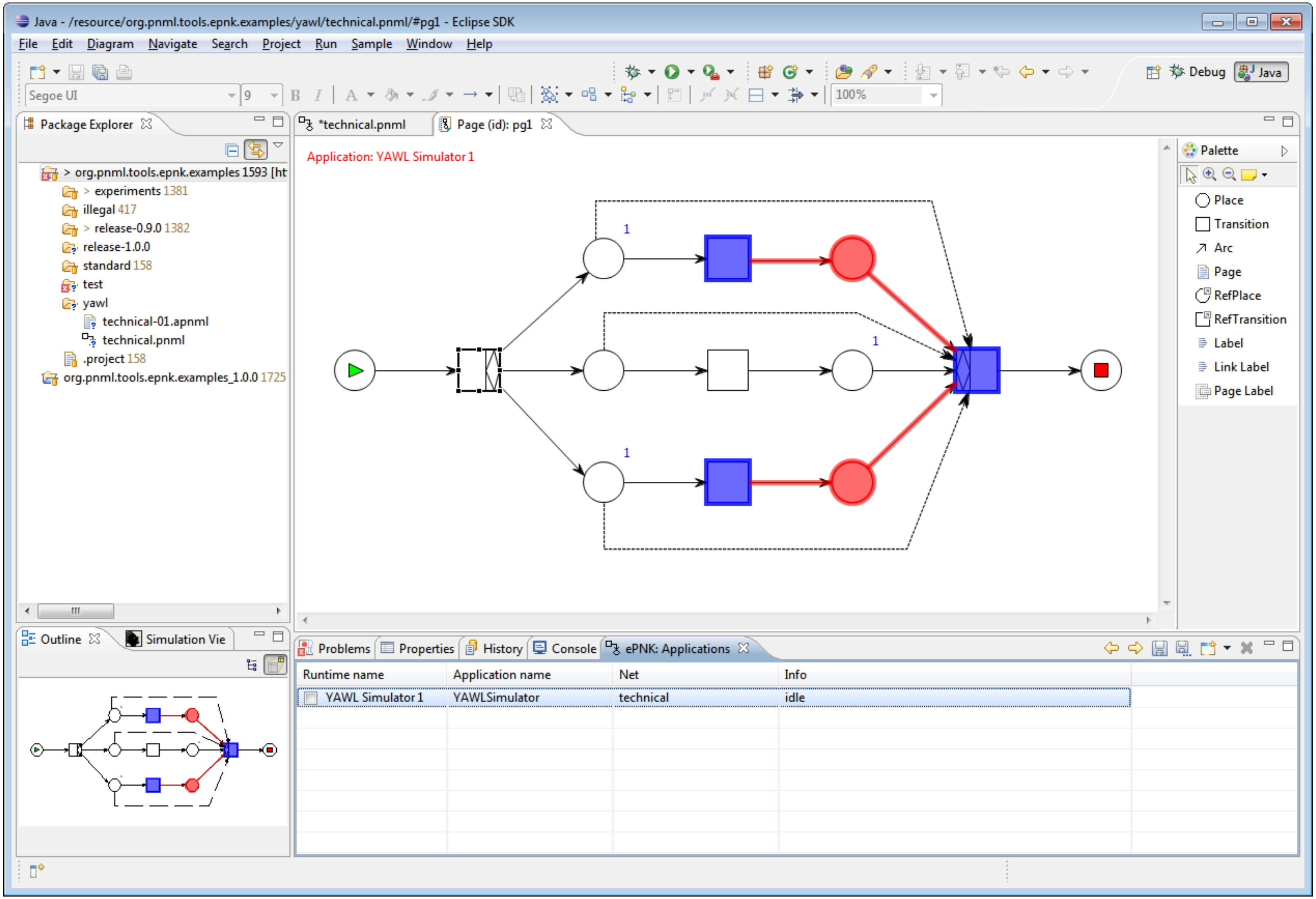


You get:

- Graphical overlays on top of the graphical editor
- The user can interact with the overlays (selecting arcs, firing transitions)
- The user can save the annotations and load them again (in the YAWL example, a firing trace)

Example: YAWL simulator

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



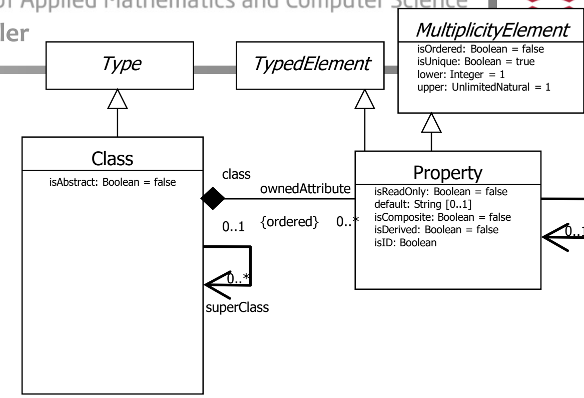
The ePNK: Material

- More information in private demo
- **ePNK: Home page**
  http://www2.compute.dtu.dk/~ekki/projects/ePNK
- Ekkart Kindler: The ePNK: A generic PNML tool - **Users' and Developers' Guide for Version** 1.0.0. IMM-Technical Report-2012-14, DTU Informatics, Kgs. Lyngby, Denmark, December 2012 (available online via ePNK home page).
- **Eclipse update site** (Indigo – Photon – 4.11): ePNK 1.2
  http://www2.compute.dtu.dk/~ekki/projects/ePNK/1.2/update/

A draft for version 1.2 is available at this course's home page (updated half way through ☹)

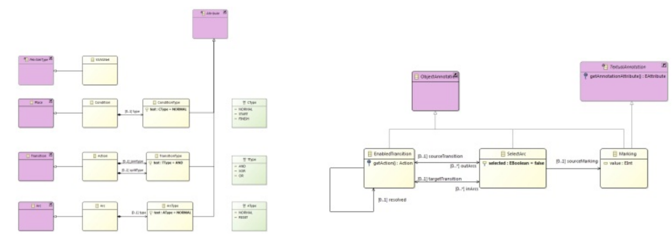
ePNK in MOF

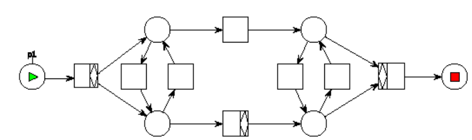
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

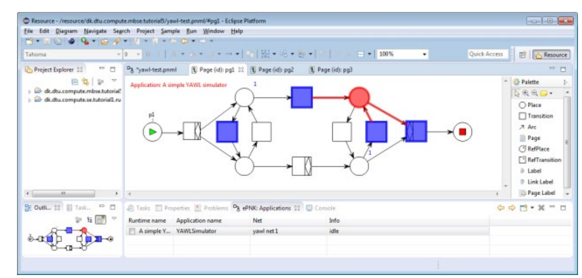


M3	Ecore (~ EMOF)
M2	ePNK and YAWL meta models + Annotation model (runtime simulator)
M1	YAWL model
M0	YAWL case (instance of a simulation – execution trace)

↑ = conforms to / is instance of







Appendix

Experience

- Agile development approach (no major design in advance)
  - In principle possible. But, you
  - need to know pitfalls of technology (unexpected implementations) well
  - manual changes must be made with good understanding of technology in order to achieve maintainability
    - this is more tricky for GMF than for EMF (but possible)

This is not in principle a problem for MBSE, but a problem with current tool support.

- Documentation missing
  - Many cool and important features of EMF/GMF are not documented
    - Guess what could be supported
    - Understand philosophy behind
    - Debug to find out details

- EMF/GMF is solid technology
  - if you know how to use and understand the philosophy behind
  - some parts are made for a very specific purpose and are not as general as suggested (ExtendedMetaData) ,

Experience

- Time effort: Altogether (up to version 0.9.0) < 5weeks
  - ca. **1 week** for making the core model and implementing core infrastructure (only EMF, generic Petri net types, XML mapping mechanism)
  - ca. **1 week** for HLPNG Petri net type, the model, its PNML-mappings and the parser for labels (Xtext)
  - ca. **½ week** for extending the PNML-mapping infrastructure so that all HLPNG features can be mapped to XML
  - ca. **½ week** for implementing the validation constraints for HLPNG (correct typing of expressions, resolution of types, …)
  - ca. **1 week** for graphical for graphical editor
  - ca. **½ week** for brushing up the graphical editor (and cleaning a bit up behind the scenes)

Part of that 1 week of debugging! 2 days my own bugs; 3 days replacing missing documentation!

As of version 0.9.2 with some extra features: 8 weeks!

Code inspection

- Petri Net Type: P/T-Net plugin
  - model for P/T-nets
  - XML-mapping: 2 lines
  - manual changes in one generated class (4 lines, 2 of them for the above XML-mapping)
  - 1 OCL constraint

Implementing the complete PNTD for the hands-on project takes me about half an hour.

- Tool-specific extension: Token position plug-in
  - model for token positions
  - no XML-/PNML mapping
  - manual creation of one class (25 lines, making the “pieces” know to Eclipse)

- GMF/EMF-editor integration
  - 45 @generated NOTs

These figures refer to ePNK version 0.9.0!

Code inspection

- Petri Net Type: HLPNG plug-in
  - model for HLPNG-nets
  - Xtext grammar for concrete syntax
  - PNML-mapping: ca. 70 entries (+ Factory)
  - manual changes in generated classes: ca. 130 (mostly functionality implementing type and sort resolution functions and helpers)
  - 1 OCL constraint, and 11 constraint classes (complex constraints)

These figures refer to ePNK version 0.9.0!

Statistics

- Project contains
  - 20 eclipse plug-in projects (11 automatically generated)
  - 10 models (+ 1 grammar)
  - 125 model classes (and interfaces)
  - ca. 800 code classes
  - ca. 36.000 MLOC (> 50.000 TLOC)
  - ca. 220 “@generated NOT” tags
  - (guess < 2000 manual lines of code)

These figures refer to ePNK version 0.9.0!