The Petri Net Markup Language
theory and practice

Advanced Tutorial at Petri Nets ‘09

Lom Hillah and Ekkart Kindler

The Petri Net Markup Language

theory and practice

Advanced Tutorial at Petri Nets '09

Lom Hillah and Ekkart Kindler

tool/framework

concepts

Schedule

14^{00} -15^{30} Ekkart Kindler

PNML “Theory”: Objectives, principles, concepts and XML syntax

15^{30} -16^{00} *Coffee break*

16^{00} -17^{30} Lom Hillah

PNML “Practice”: Making it work – the PNML Framework

The Petri Net Markup Language
theory and practice

Ekkart Kindler

Denmark's Technical University

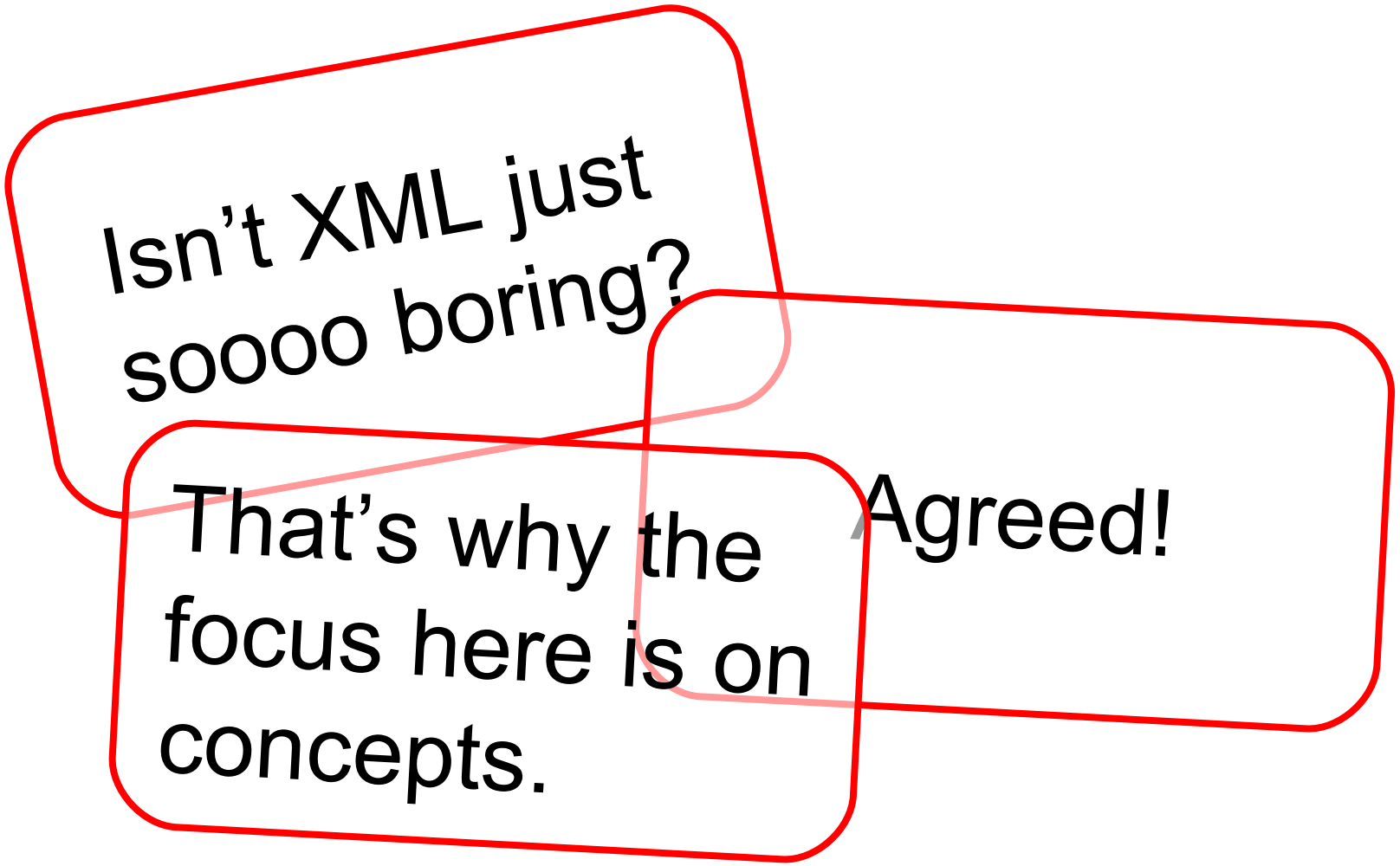
DTU Informatics

PNML in a nutshell

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

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

Note that Part 2 and Part 3 are not international standards yet.


Isn't XML just soooo boring?
That's why the focus here is on concepts.
Agreed!

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 behind/of PNML.

This tutorial focuses more on the concepts; the XML-syntax comes on the side.

ISO/IEC 15909 and PNML

- ISO/IEC 15909-1 defines High-level Petri Nets (2004)
- ISO/IEC 15909-2 defines a transfer format for High-level Petri Nets based on PNML (currently under FDIS ballot)
- ISO/IEC 15909-3 will define net extensions and the underlying concepts of PNML

Note that this is not (yet) an international standard.

PNML in a nutshell

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

Why do we need a standard?

What’s the problem?

Why standards?

- Boost use of tools and interchange of Petri nets among tools
- Increase visibility
- Industry just “loves” standards

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.

Principle

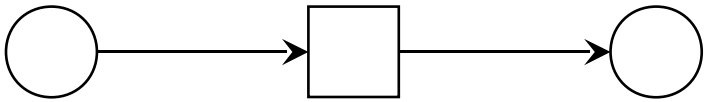
PNML is an XML-based transfer format for "all kinds" of Petri nets

- flexible and universal
- clear and unequivocal
- compatible

Flexibility also concerns the process.

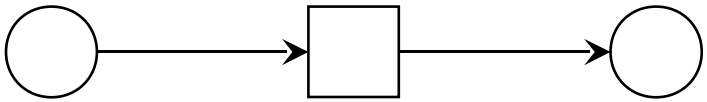
A first example


```
<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

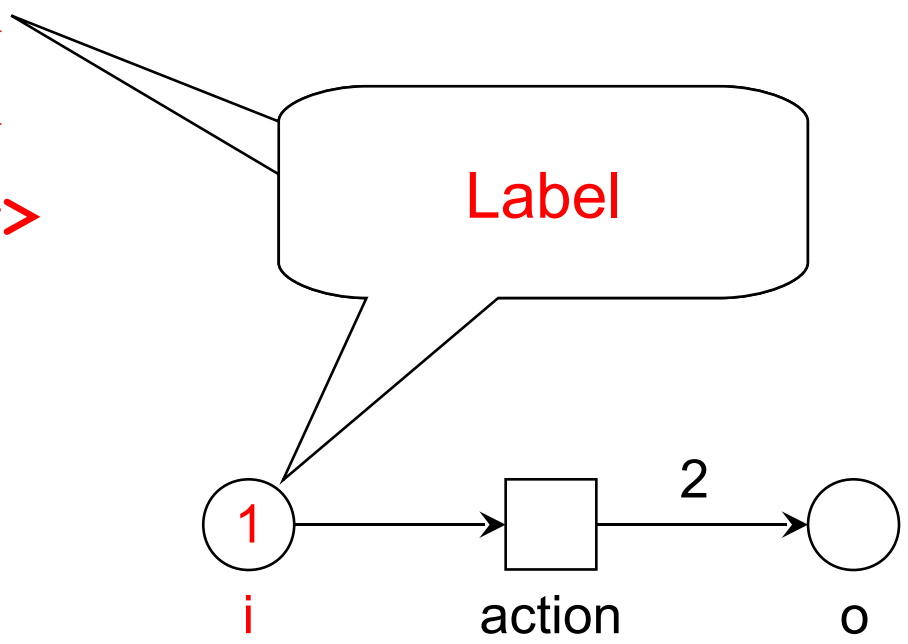

```
<pnml xmlns="http://www.pnml.org/...">
  <net id="n1" type="...">
    ...
    <place id="p1"/>
    <arc id="a1" source="p1" target="t1"/>
    <transition id="t1"/>
    <arc id="a2" source="t1" target="p2"/>
    <place id="p2"/>
    ...
  </net>
</pnml>
```



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.



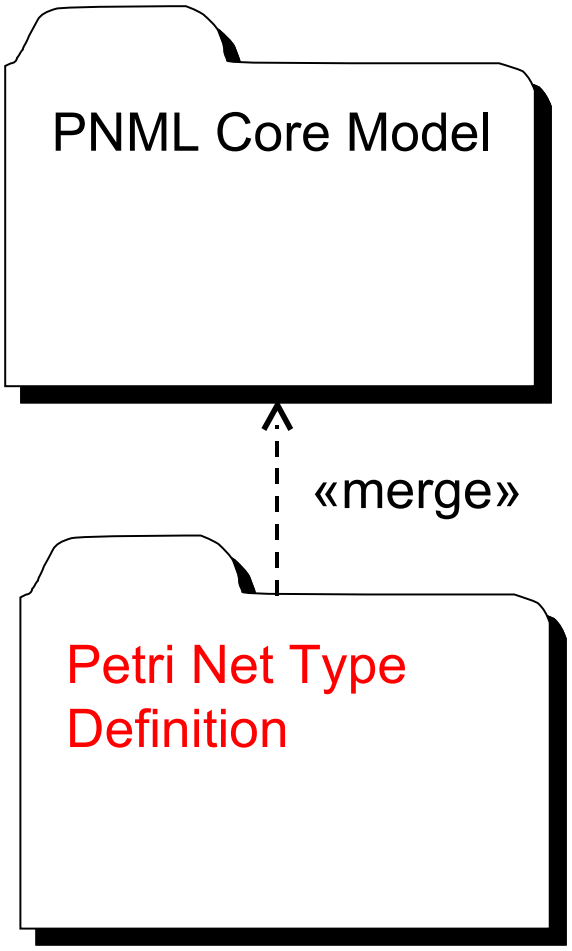
Basic Idea

„All kinds“ of Petri nets can be represented by

- places
- transitions, and
- arcs

along with some

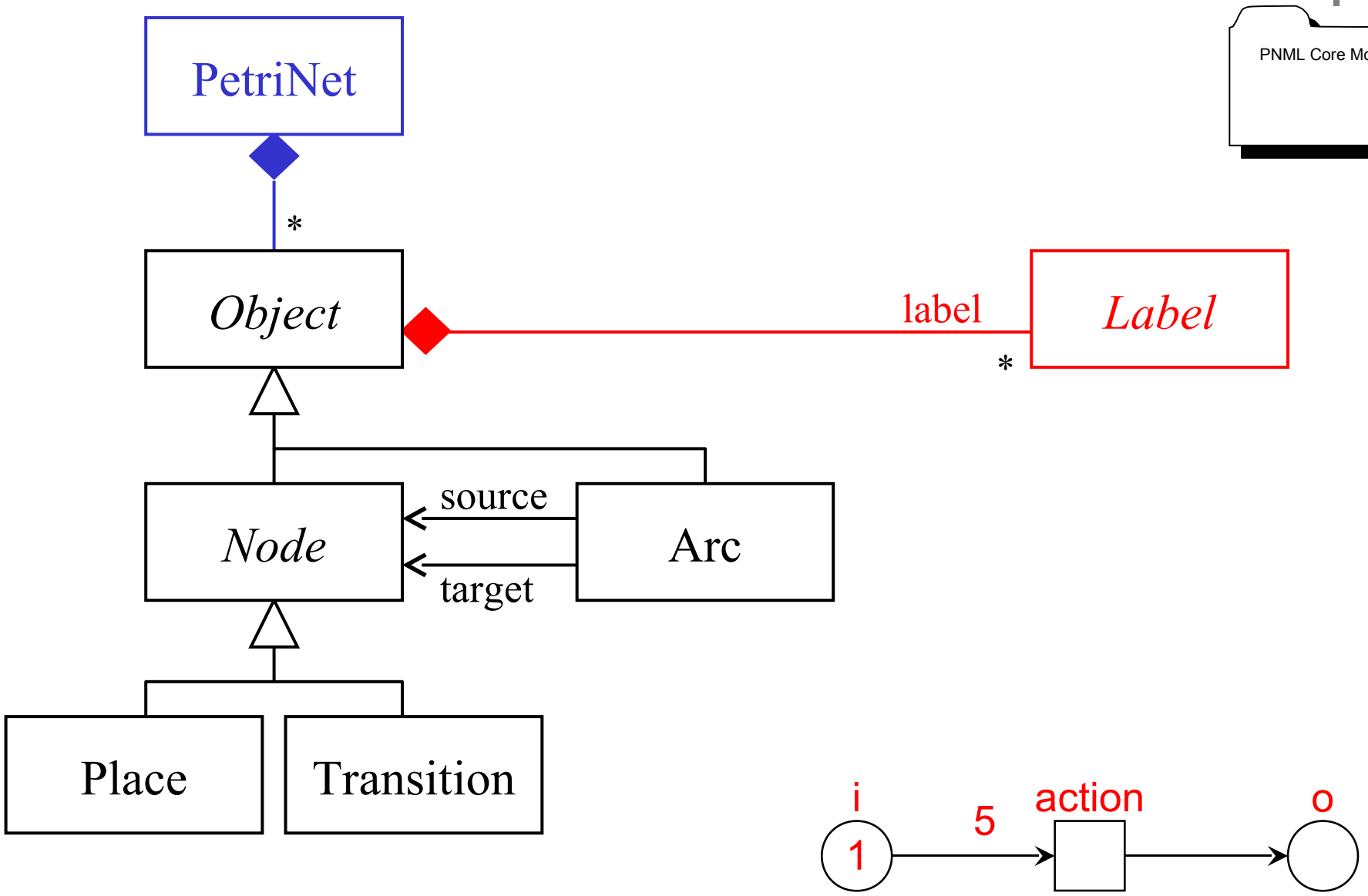
- labels



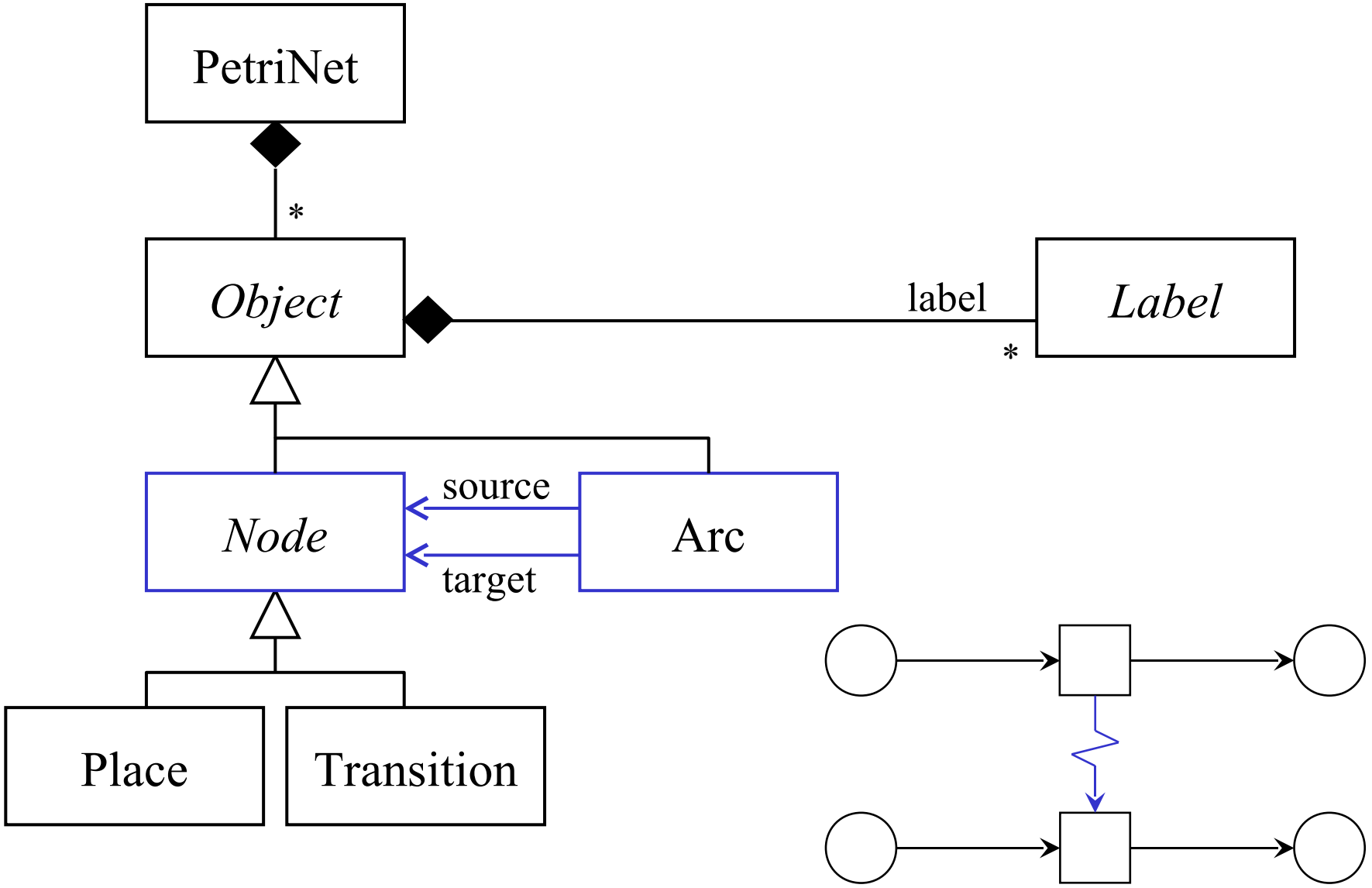
Outline

- Introduction and Motivation
- Basic concepts
  - Core model
  - Type model
- Mapping to XML
- More details
- Extensions and Current & Future Work
- Conclusions

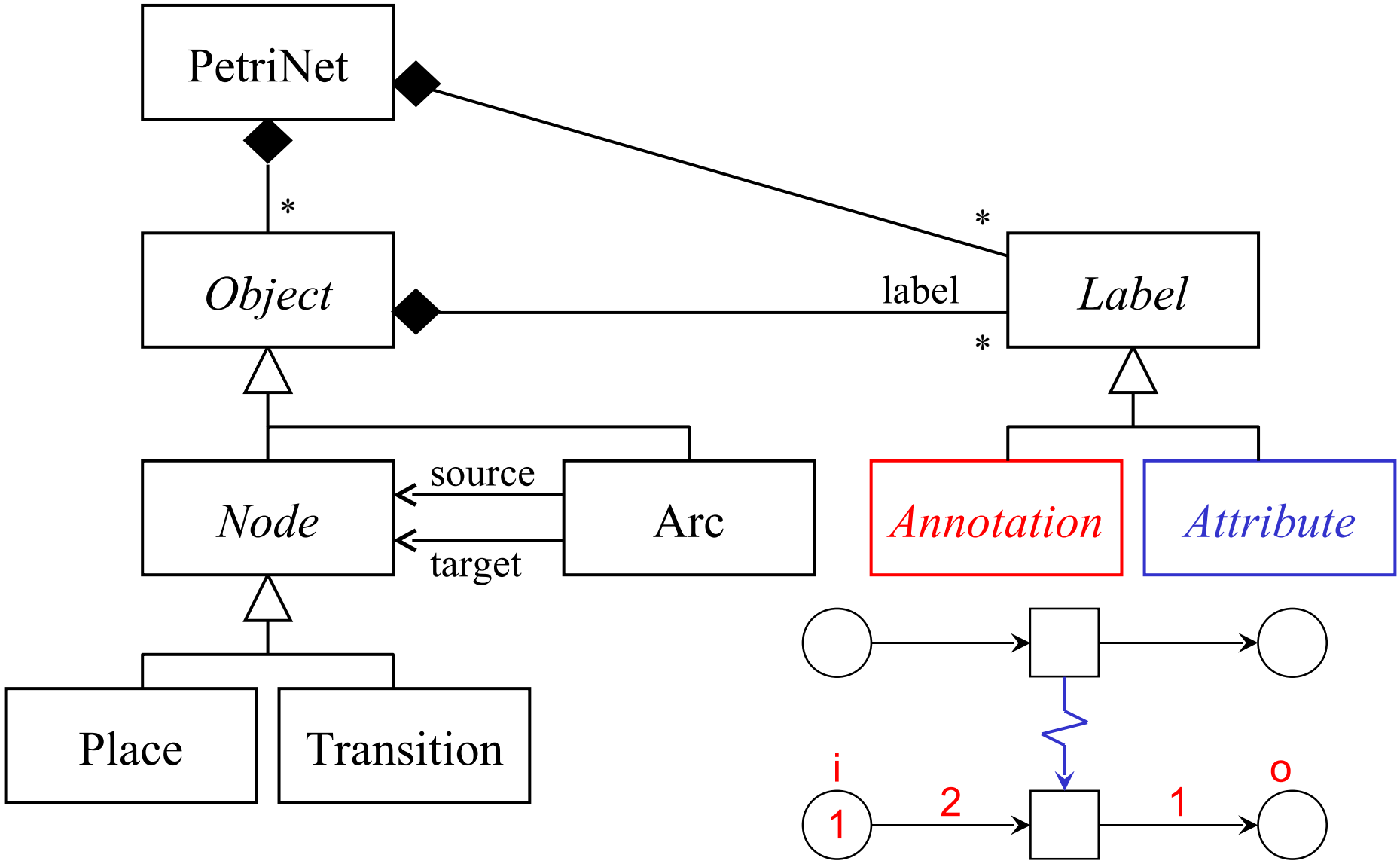
Core Model (overview)



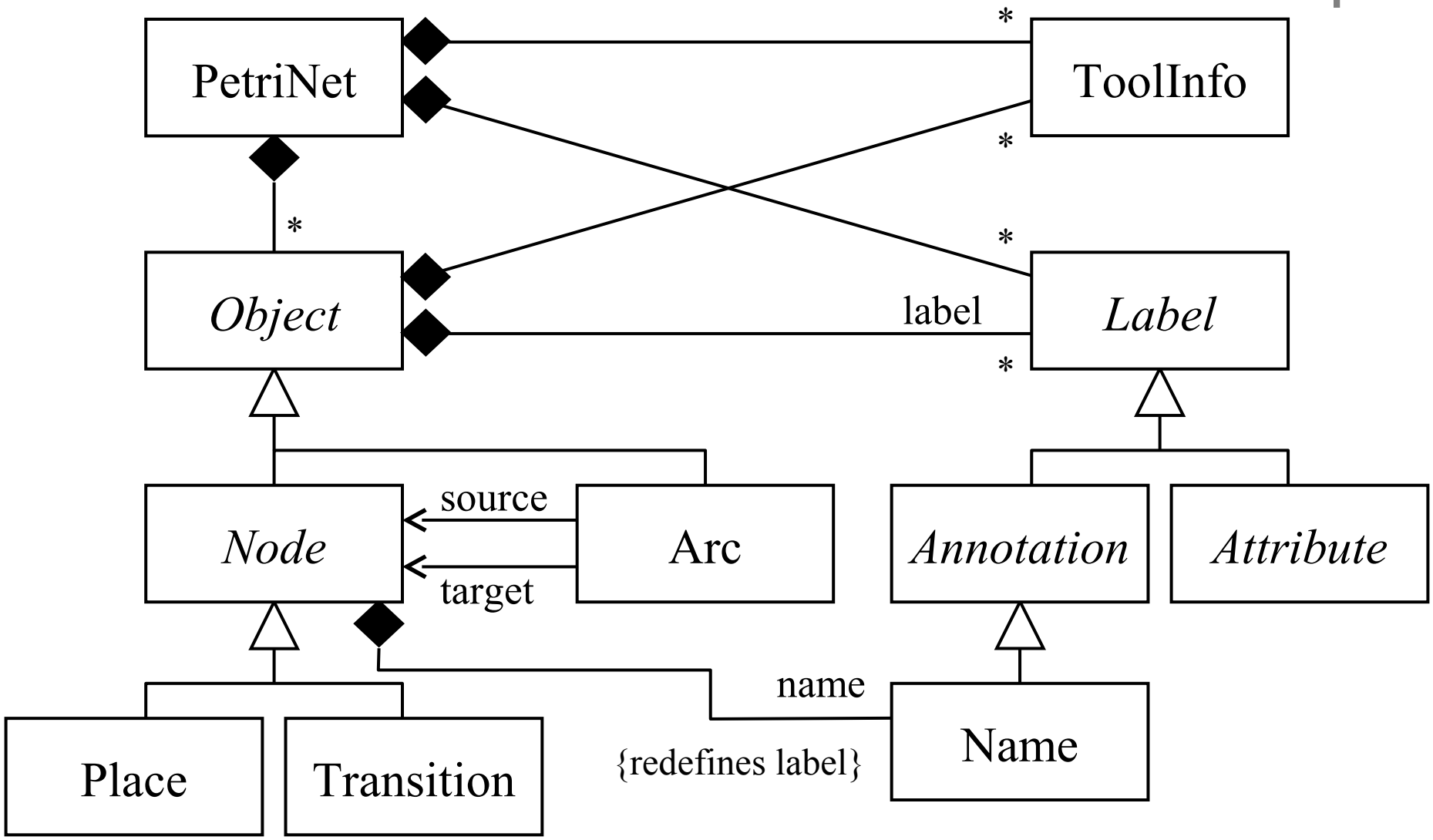
Core Model (overview)



Core Model (overview)

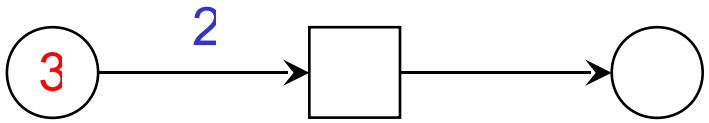


Core Model (overview)

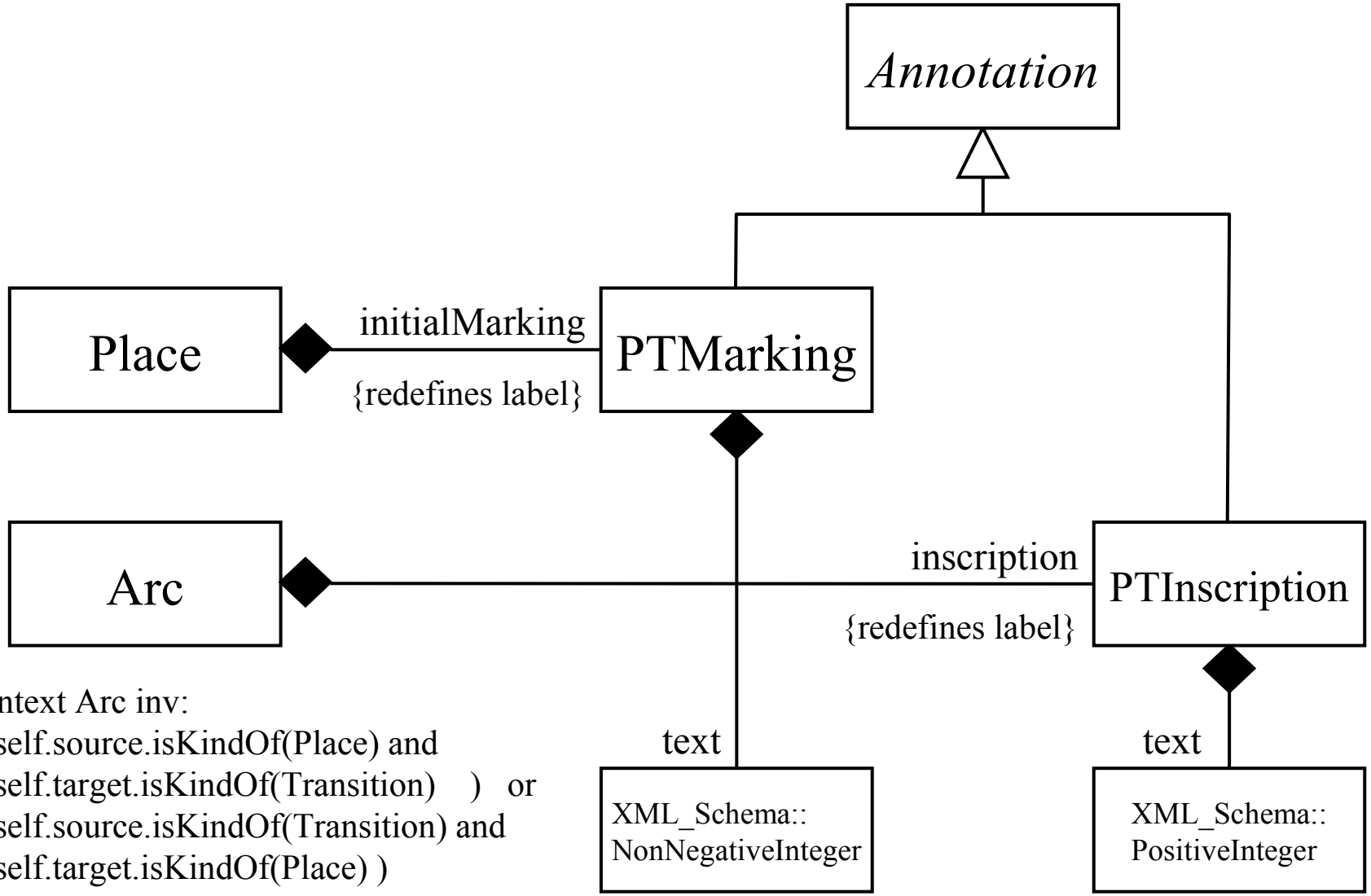


Type Definition: PT-Net

Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler

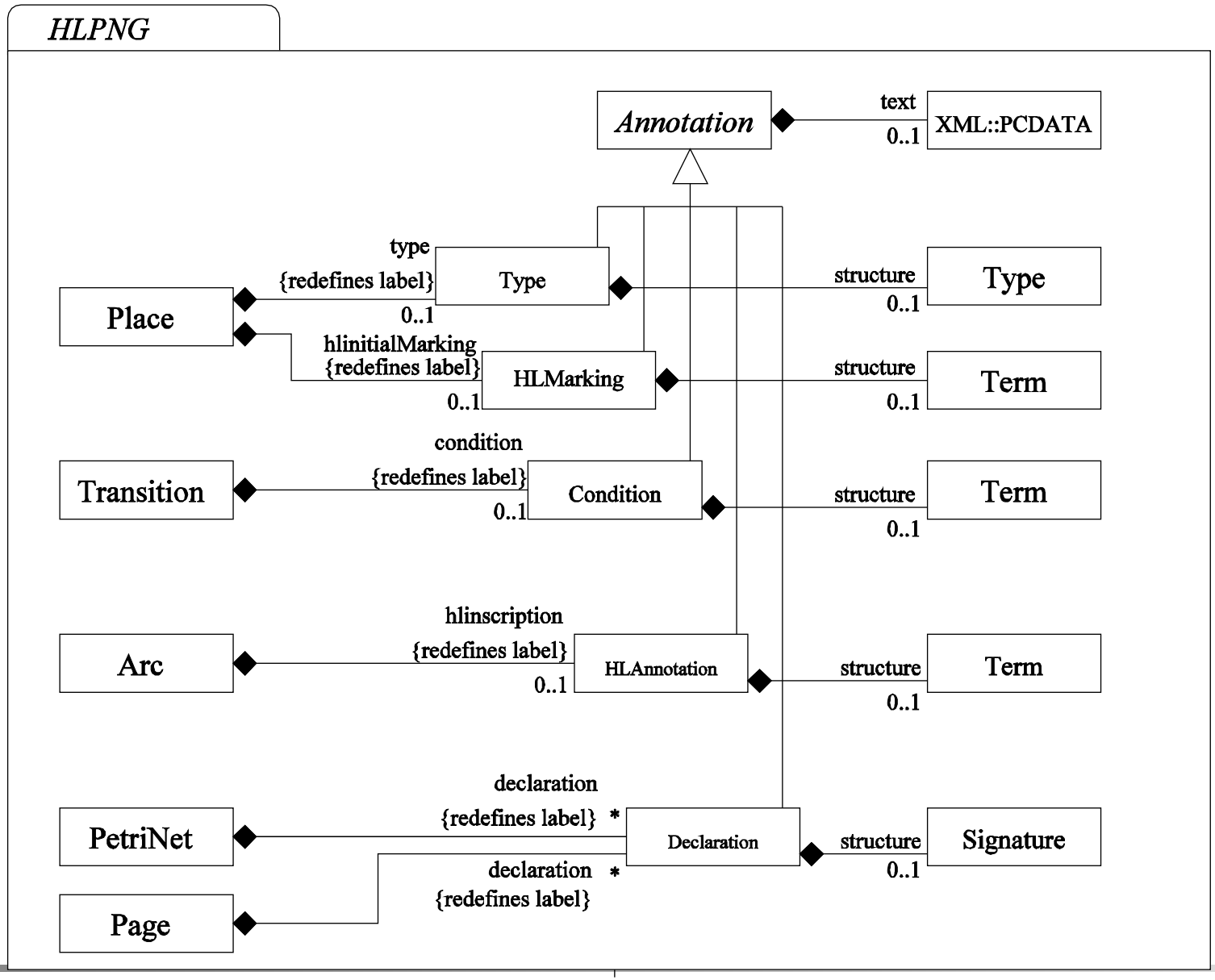


Type Definition: PT-Net



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)

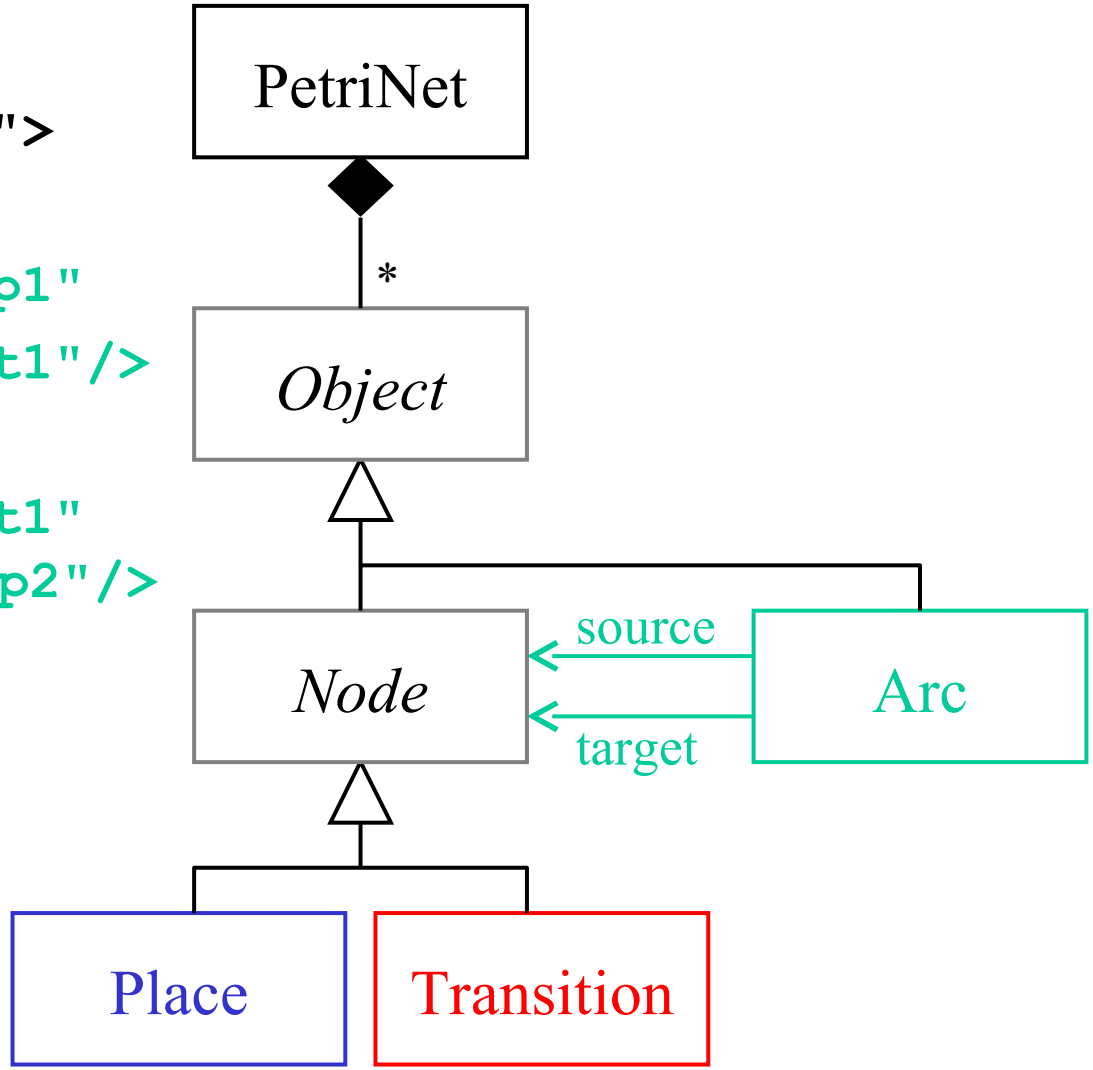


Outline

- Introduction and Motivation
  - Overview
  - Principles
  - Basic idea
- Basic concepts
  - Core model
  - Type model
- Mapping to XML
- More details
- Extensions and Current & Future Work
- Conclusions

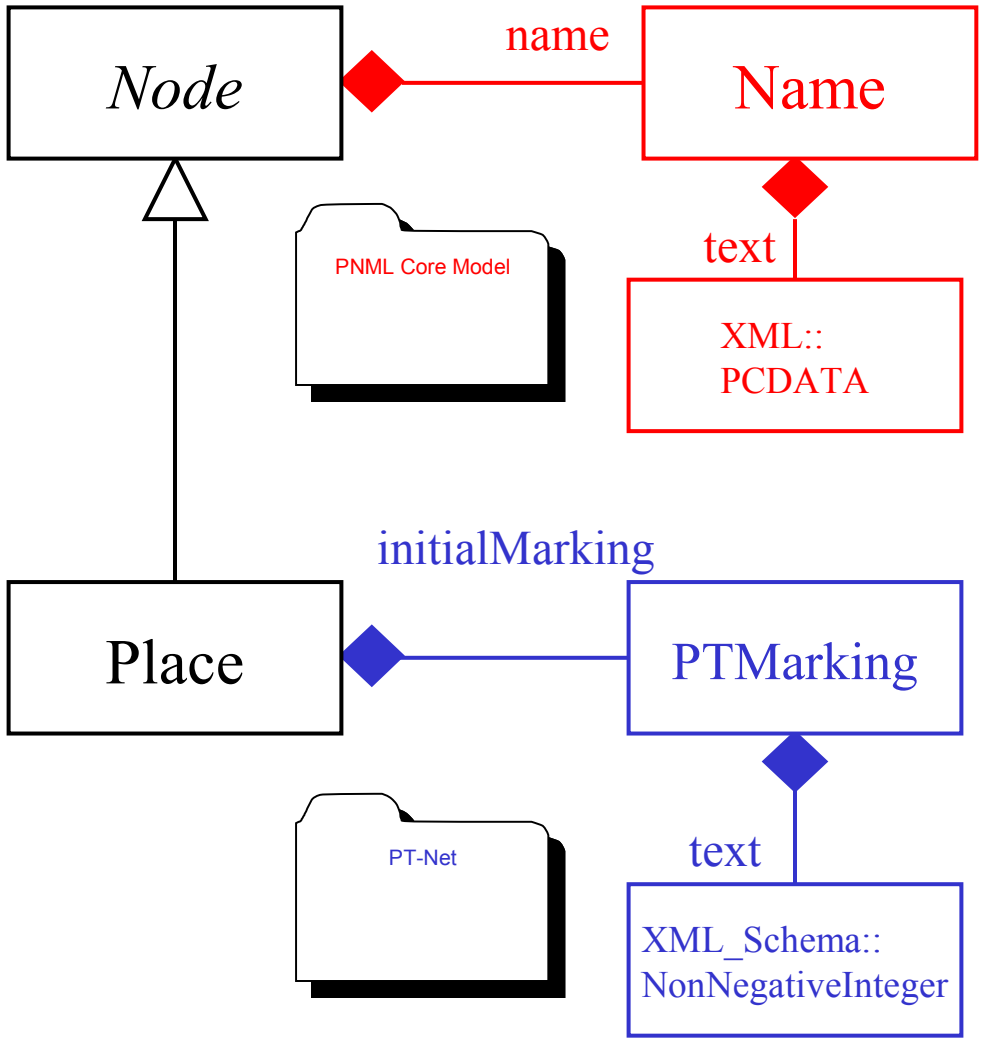
Core Model in XML


```
<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

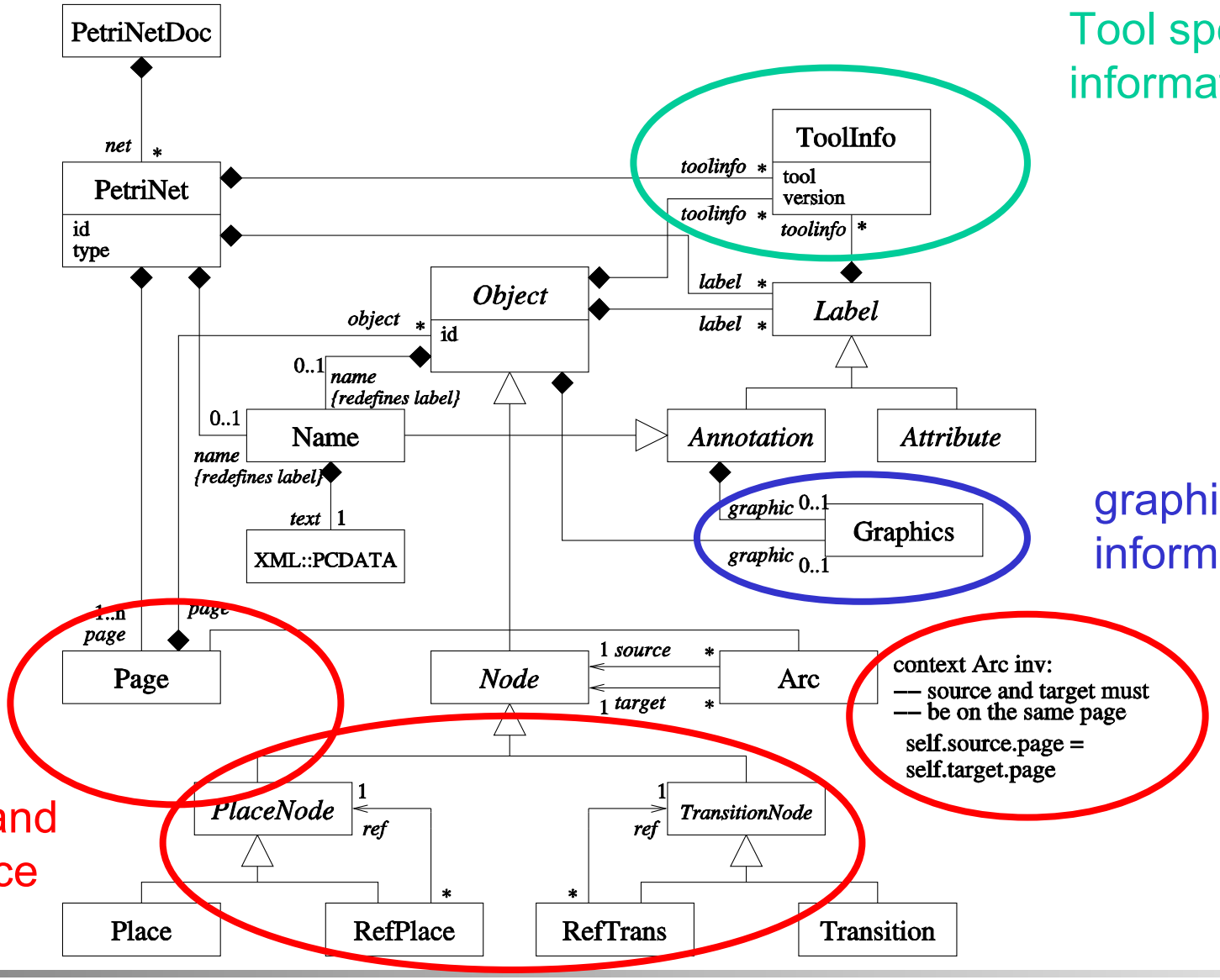

```
...
  <place id="p1">
    <name>
      <text>i</text>
    </name>
    <initialMarking>
      <text>1</text>
    </initialMarking>
  </place>
...
```



Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
  - Toolspecific information
  - Pages, reference nodes
  - Graphics
  - High-level nets (overview)
  - Reading the standard
- Extensions and Current & Future Work
- Conclusions

PNML Core Model



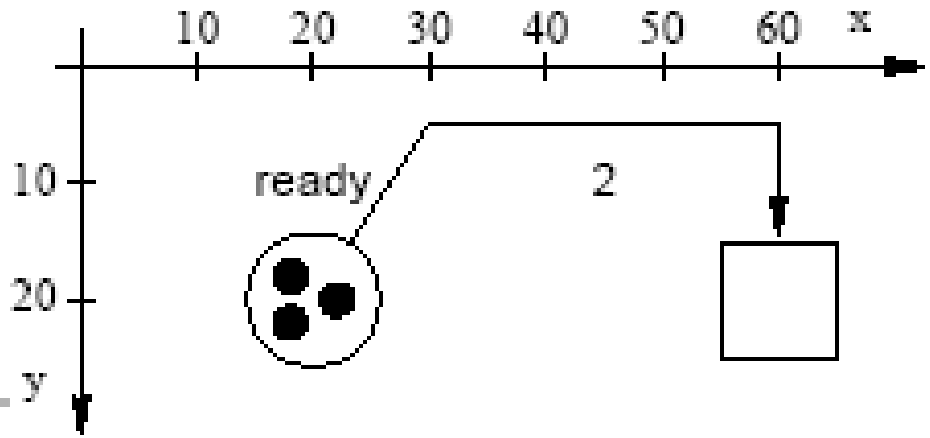
Tool specific information

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>
```



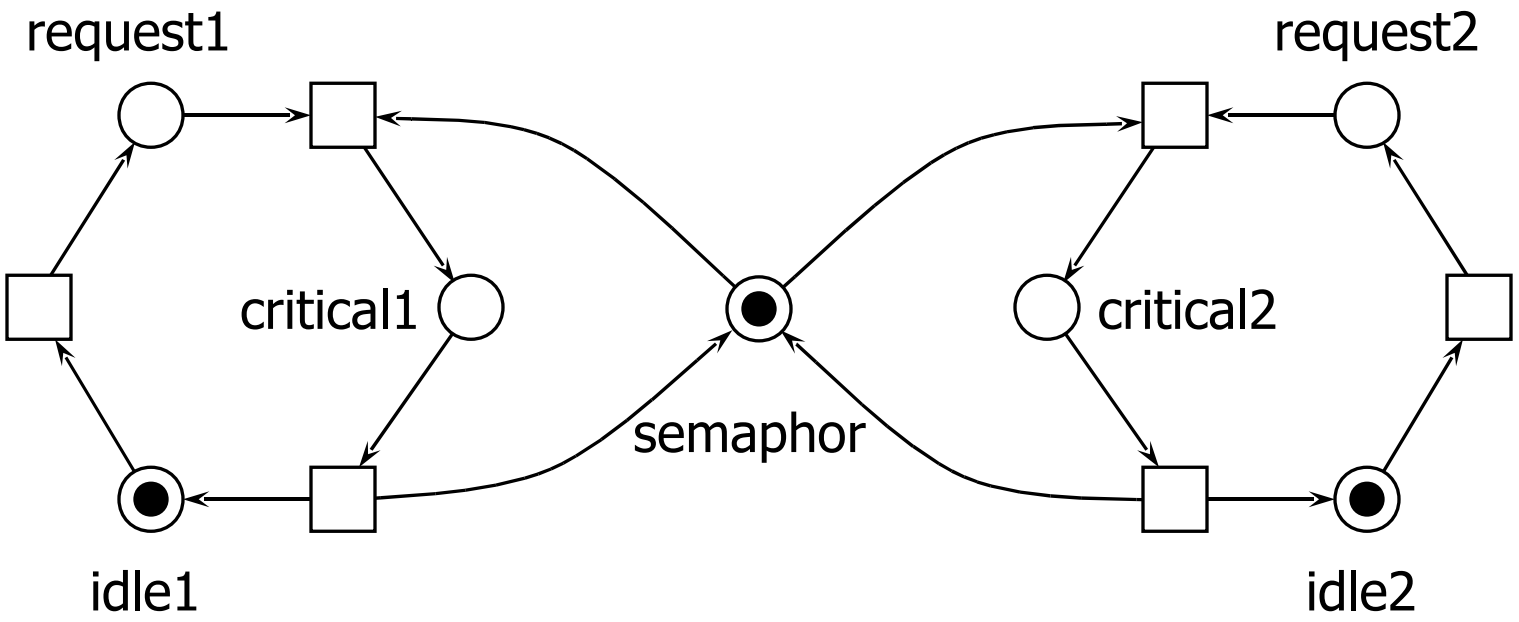
Tool specific information

Tool specific information can be used to store any additional information with any Petri net element, where a tool deems that necessary.

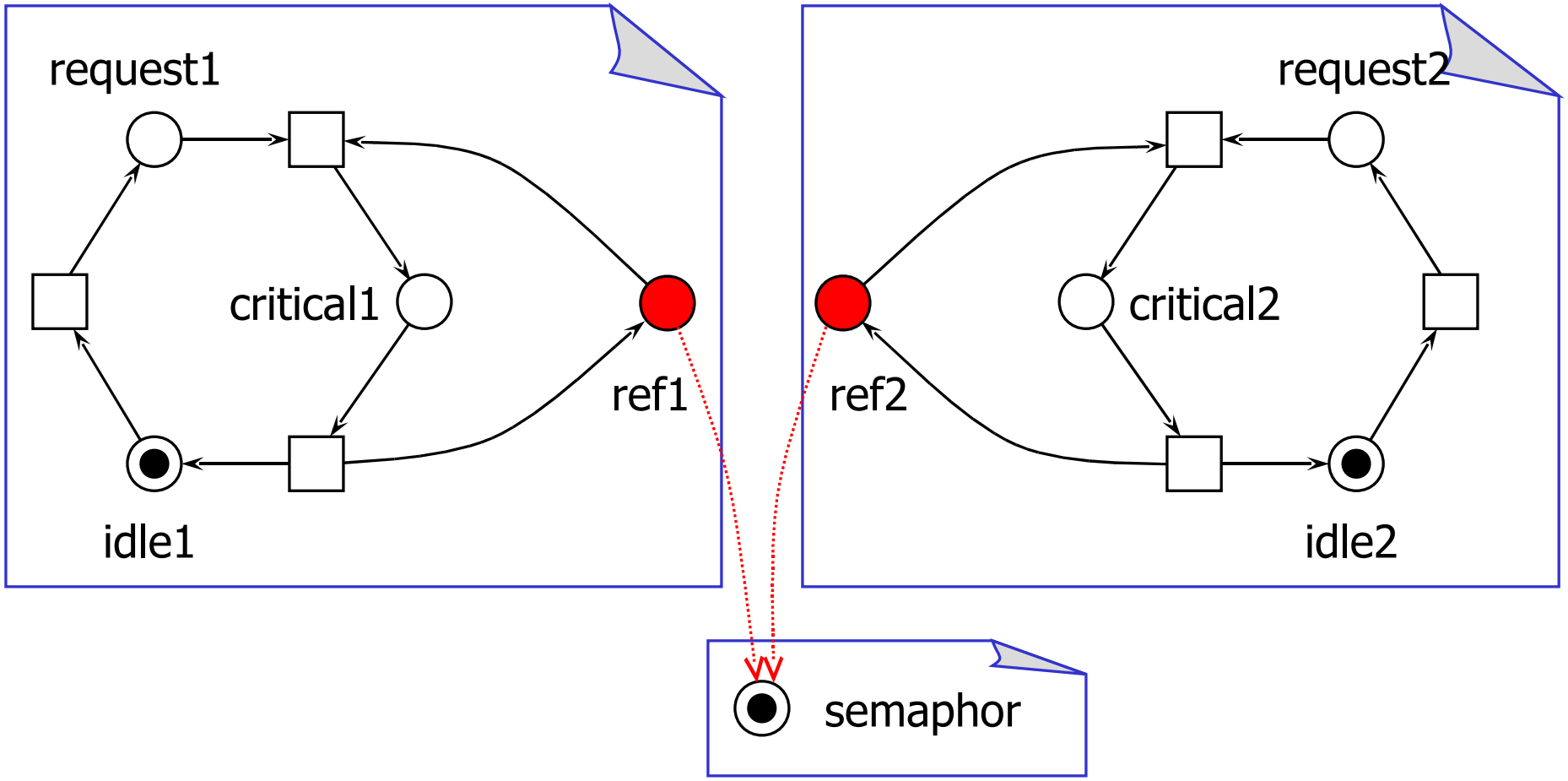
Recommendation:

- Other tools should not touch it (as long as the respective element is not deleted)
- Contents should be local!
- No conditions on contents, except that it be well-formed XML

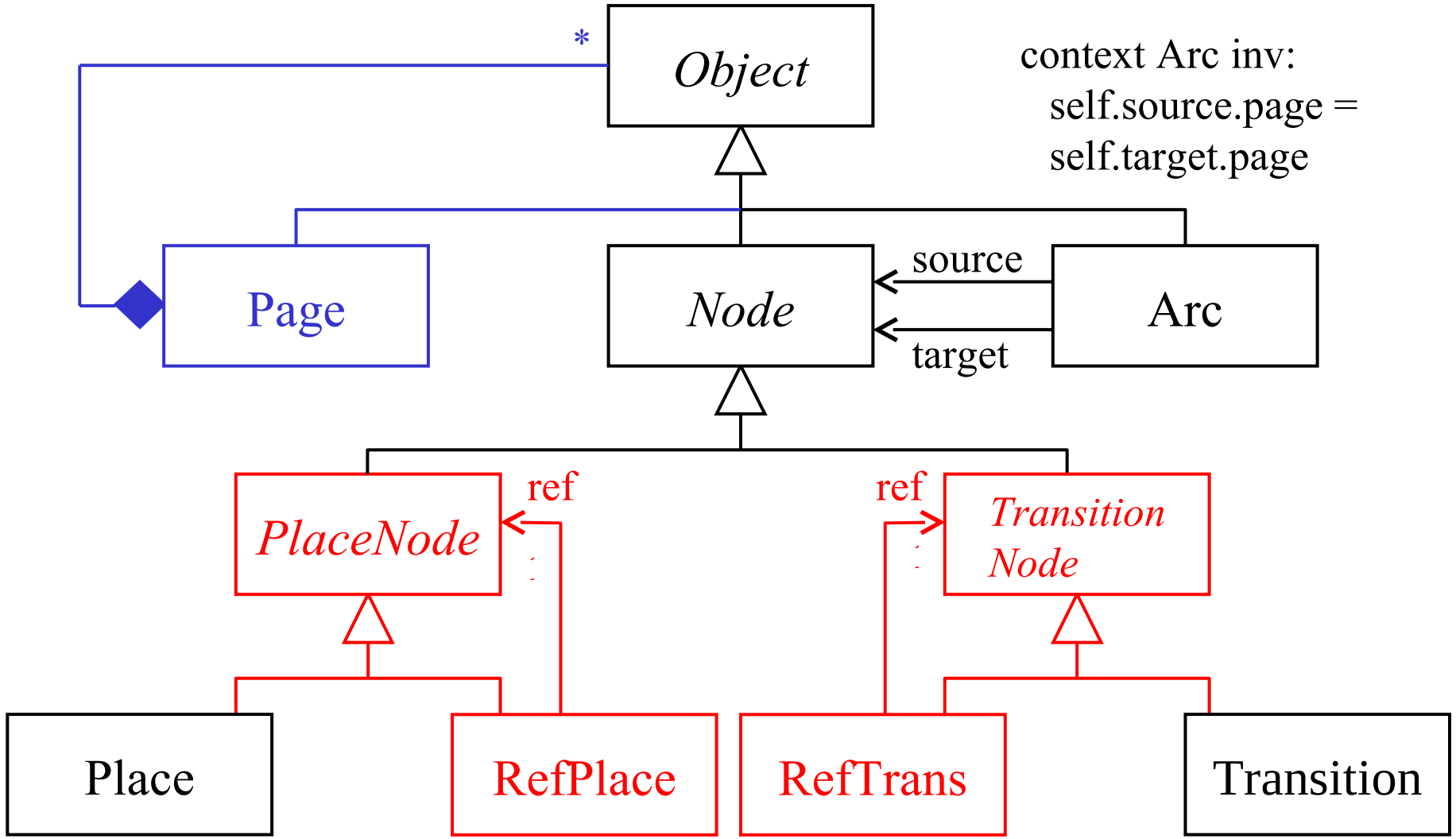
A “large” Petri net



Pages and reference nodes



Pages and references



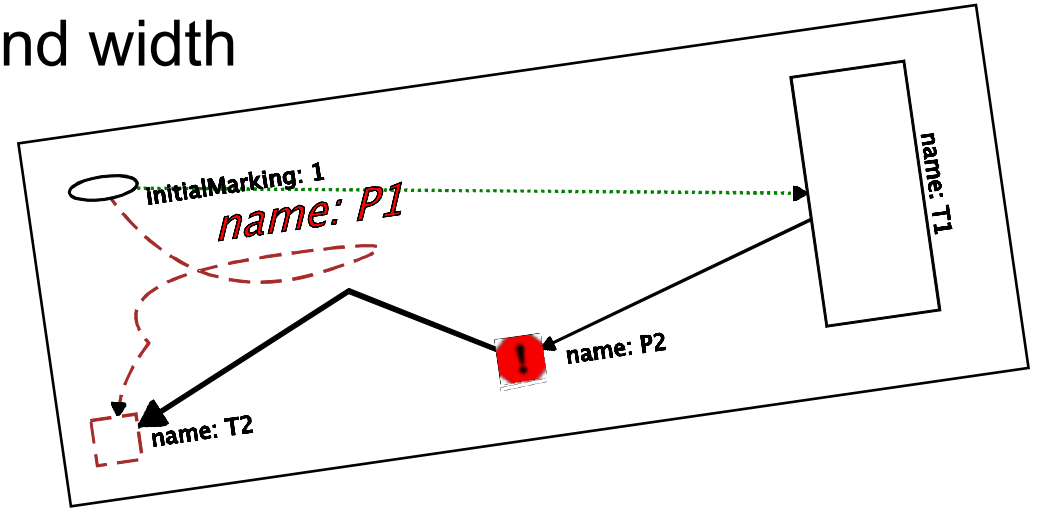
Always at least one page!


```
<pnml xmlns="http:...">
 <net id="n1" type="...">
  <page id="pg1">
   <place id="p1"/>
   <arc id="a1" source="p1"
                target="t1"/>
   <transition id="t1"/>
   <arc id="a2" source="t1"
                target="p2"/>
   <place id="p2"/>
  </page>
 </net>
</pnml>
```

Note that, every Petri net has at least one page in which the other objects are contained!

Graphical information

- nodes and pages
  - position and size
  - colors, line styles and width
  - images
- arcs
  - intermediate points
  - color, line styles and width
- annotations
  - position (offset)
  - font and size
  - …



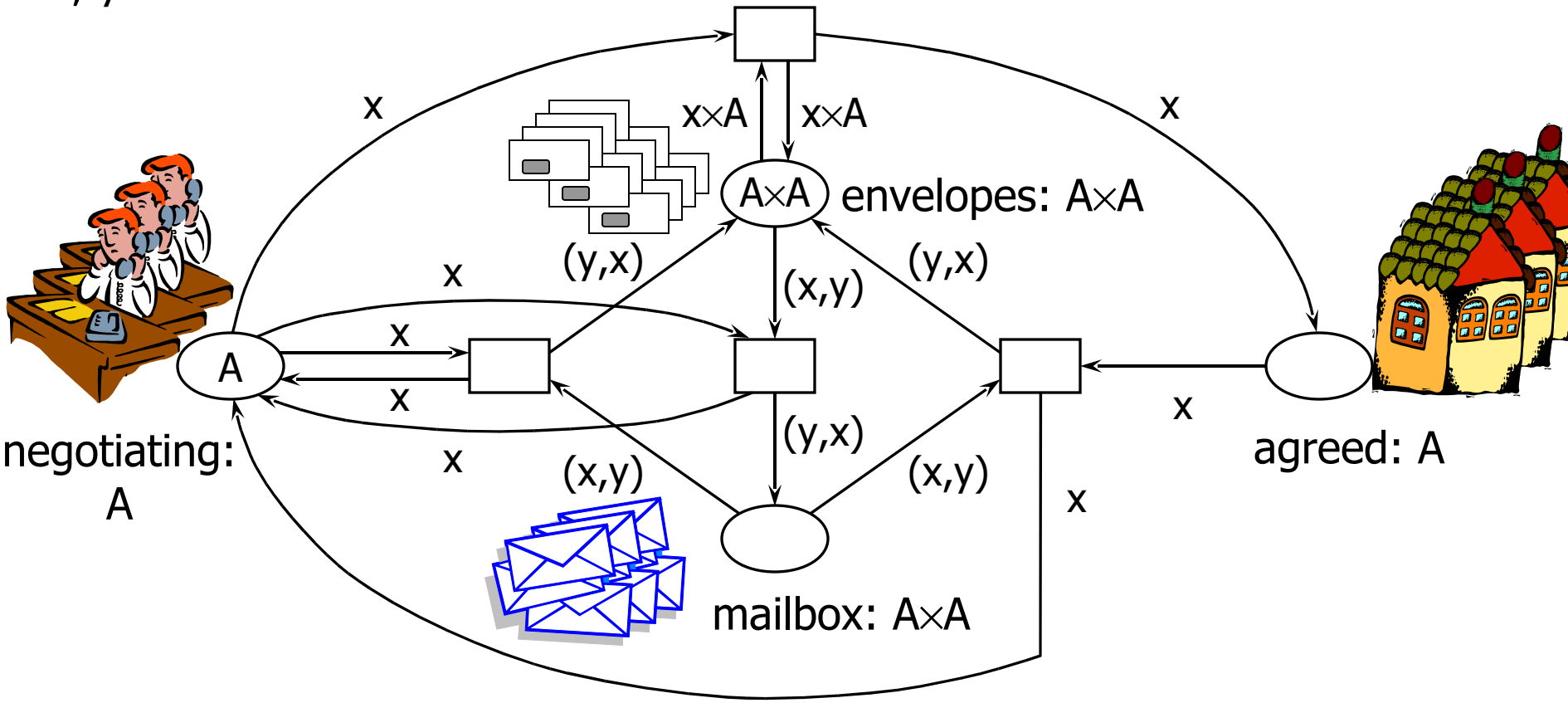
Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
  - Pages, reference nodes
  - Graphics
  - High-level nets (overview)
  - Reading the standard
- Extensions and Current & Future Work
  - Modularity
  - Type definitions
- Conclusions

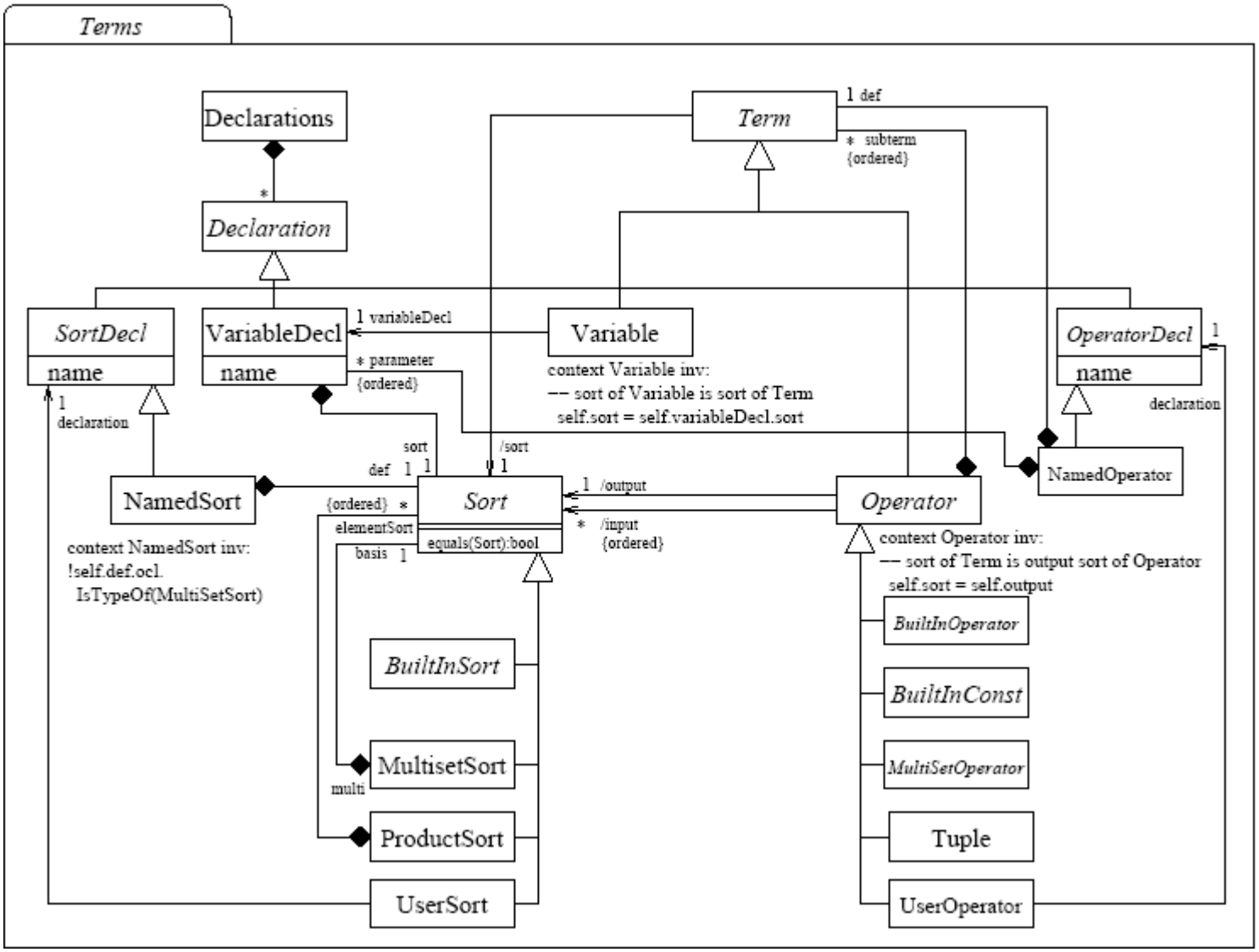
High-level nets: Example

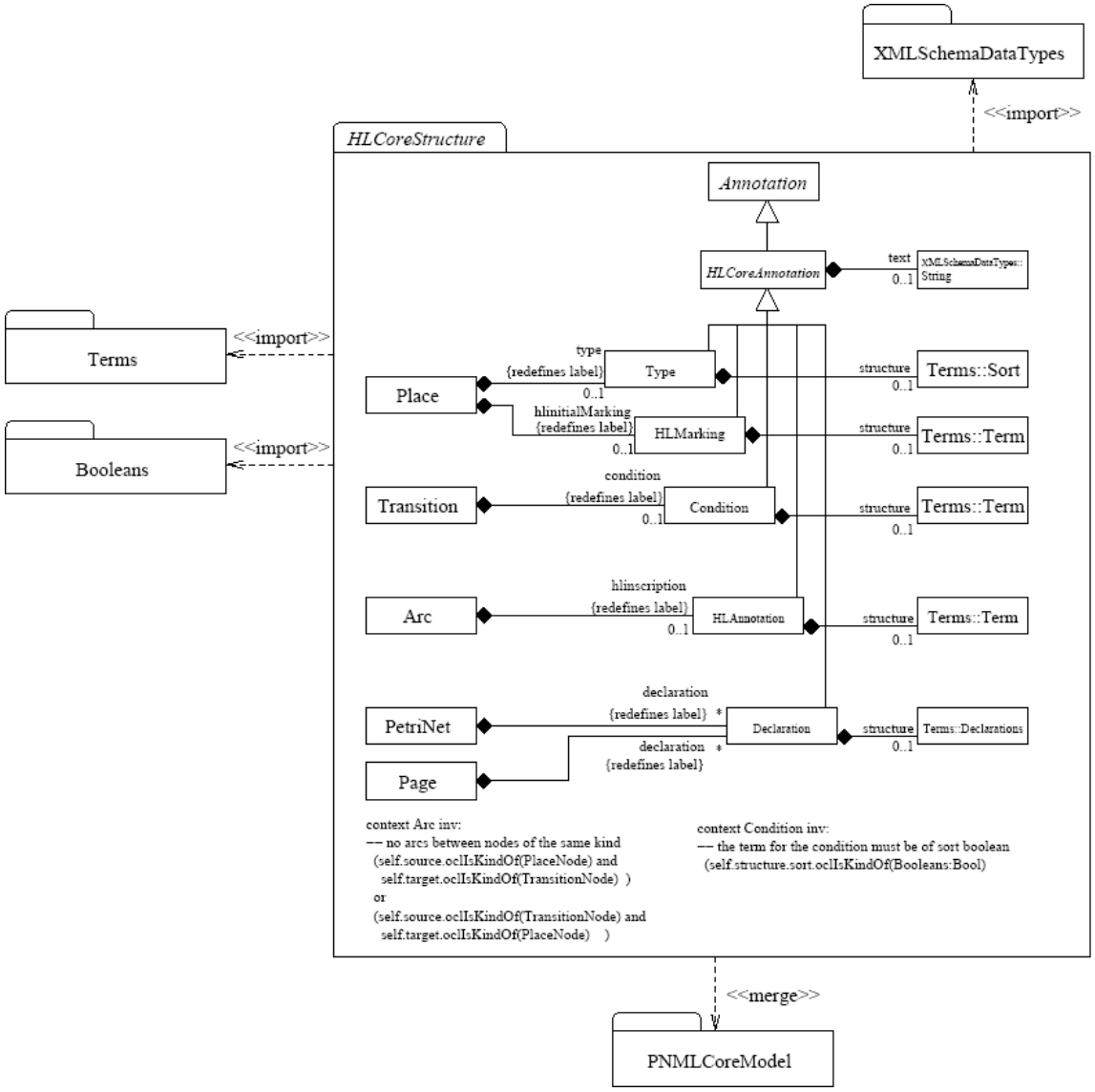
Sorts: A

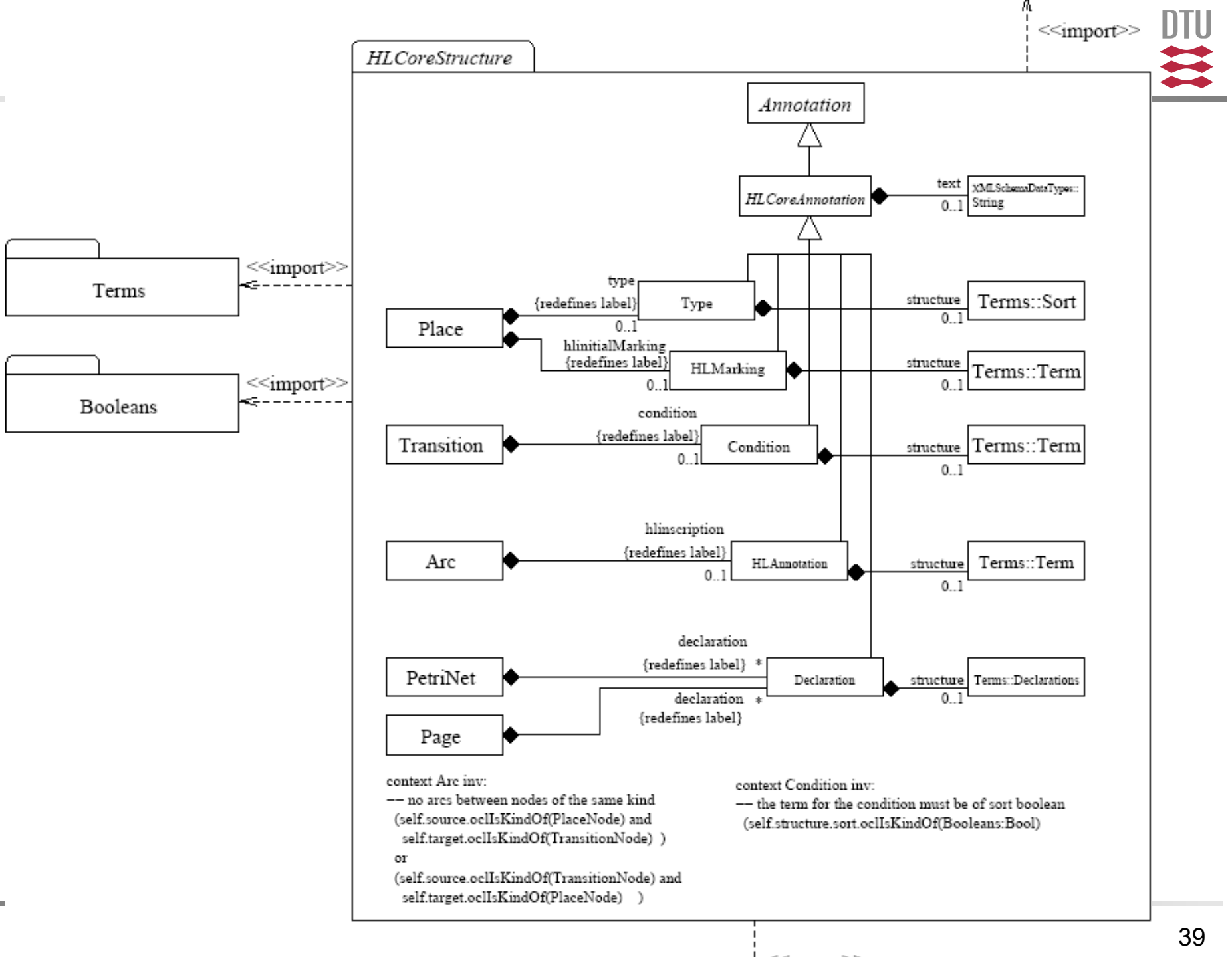
Variables:
x, y: A



Sorts, operators, terms




XMLSchemaDataTypes
<<import>>
HLCoreStructure
Annotation
HLCoreAnnotation
text
0..1
XMLSchemaDataTypes::String
Terms
<<import>>
Booleans
<<import>>
type
{redefines label}
0..1
Type
structure
0..1
Terms::Sort
Place
hlinitialMarking
{redefines label}
0..1
HLMarking
structure
0..1
Terms::Term
condition
{redefines label}
0..1
Transition
Condition
structure
0..1
Terms::Term
hlinscription
{redefines label}
0..1
Arc
HLAnnotation
structure
0..1
Terms::Term
declaration
{redefines label}
*
PetriNet
Declaration
structure
0..1
Terms::Declarations
declaration
{redefines label}
*
Page
context Arc inv:
-- no arcs between nodes of the same kind
(self.source.oclIsKindOf(PlaceNode) and
self.target.oclIsKindOf(TransitionNode))
or
(self.source.oclIsKindOf(TransitionNode) and
self.target.oclIsKindOf(PlaceNode))
context Condition inv:
-- the term for the condition must be of sort boolean
(self.structure.sort.oclIsKindOf(Booleans:Bool)
<<merge>>
PNMLCoreModel


HLCoreStructure
<<import>>
Terms
<<import>>
Booleans
<<import>>
Annotation
HLCoreAnnotation
text
0..1
XMLSchemaDataTypes::String
Place
type
{redefines label}
0..1
Type
structure
0..1
Terms::Sort
hlinitialMarking
{redefines label}
0..1
HLMarking
structure
0..1
Terms::Term
Transition
condition
{redefines label}
0..1
Condition
structure
0..1
Terms::Term
Arc
hlinscription
{redefines label}
0..1
HLAnnotation
structure
0..1
Terms::Term
PetriNet
declaration
{redefines label} *
Declaration
structure
0..1
Terms::Declarations
Page
declaration *
{redefines label}
context Arc inv:
-- no arcs between nodes of the same kind
(self.source.oclIsKindOf(PlaceNode) and
self.target.oclIsKindOf(TransitionNode))
or
(self.source.oclIsKindOf(TransitionNode) and
self.target.oclIsKindOf(PlaceNode))
context Condition inv:
-- the term for the condition must be of sort boolean
(self.structure.sort.oclIsKindOf(Booleans:Bool)

HLPN: Built-in types

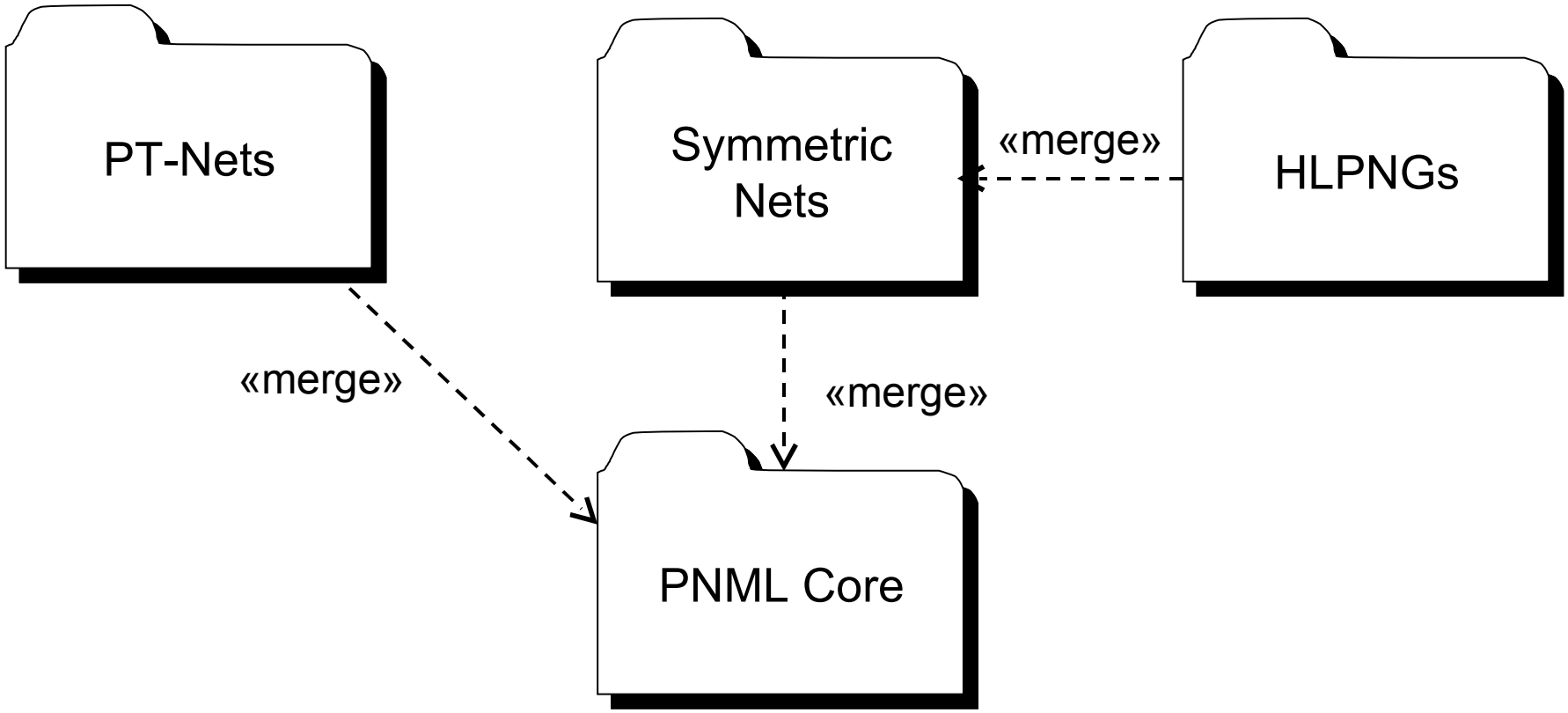
Large parts of the standard (> 1/3 of main part) deal with the definition of built-in sorts and operators:

- Dots
- Booleans
- Products
- Multisets
- Various finite domains (for Symmetric Nets)

- Naturals, positive integers, integers
- Strings
- Lists
- User definable sorts and operations

Overview of Petri net types

ISO/IEC 15909-2: defined net types and their relation



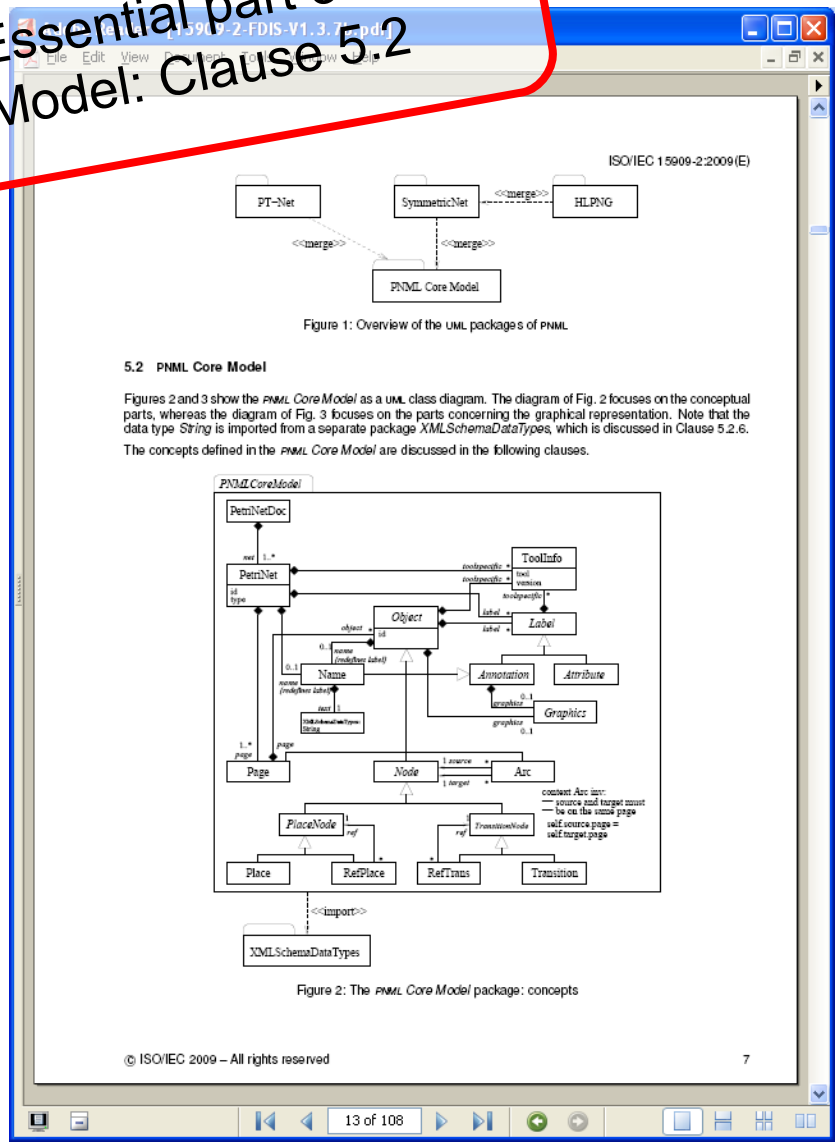
Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
  - Pages, reference nodes
  - Graphics
  - PNTD: High-level net (overview)
  - Reading the standard
- Extensions and Current & Future Work
  - Modularity
  - Type definitions
- Conclusions

Concepts

Essential part on Core Model: Clause 5.2

Graphical Features: Clause 5.2.4



ISO/IEC 15909-2:2009(E)

Figure 1: Overview of the UML packages of PNML

5.2 PNML Core Model

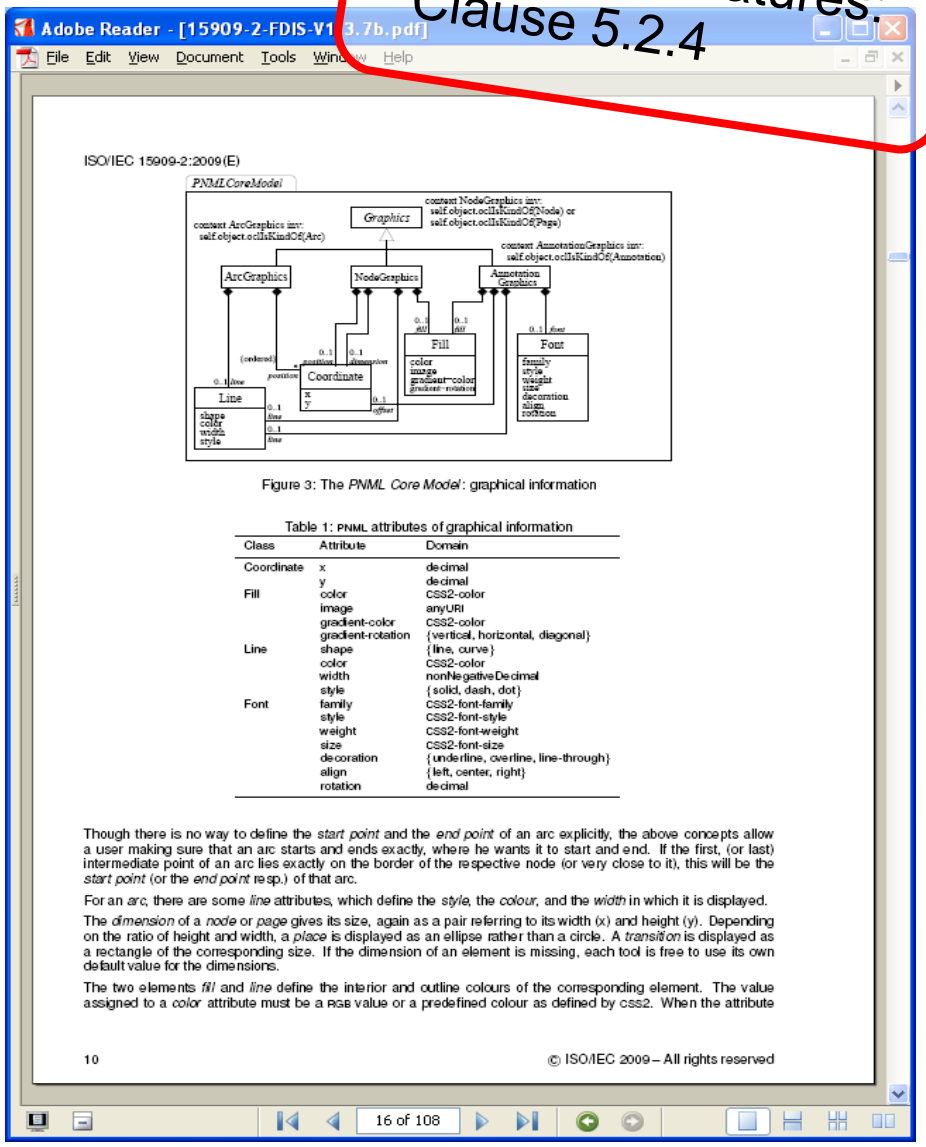
Figures 2 and 3 show the *PNML Core Model* as a UML class diagram. The diagram of Fig. 2 focuses on the conceptual parts, whereas the diagram of Fig. 3 focuses on the parts concerning the graphical representation. Note that the data type *String* is imported from a separate package *XMLSchemaDataTypes*, which is discussed in Clause 5.2.6.

The concepts defined in the *PNML Core Model* are discussed in the following clauses.

Figure 2: The *PNML Core Model* package: concepts

© ISO/IEC 2009 – All rights reserved

13 of 108



ISO/IEC 15909-2:2009(E)

Figure 3: The *PNML Core Model*: graphical information

Table 1: PNML attributes of graphical information

Class	Attribute	Domain
Coordinate	x	decimal
	y	decimal
Fill	color	CSS2-color
	image	anyURI
	gradient-color	CSS2-color
	gradient-rotation	{vertical, horizontal, diagonal}
Line	shape	{line, curve}
	color	CSS2-color
	width	nonNegativeDecimal
	style	{solid, dash, dot}
Font	family	CSS2-font-family
	style	CSS2-font-style
	weight	CSS2-font-weight
	size	CSS2-font-size
	decoration	{underline, overline, line-through}
	align	{left, center, right}
	rotation	decimal

Though there is no way to define the *start point* and the *end point* of an arc explicitly, the above concepts allow a user making sure that an arc starts and ends exactly, where he wants it to start and end. If the first, (or last) intermediate point of an arc lies exactly on the border of the respective node (or very close to it), this will be the *start point* (or the *end point* resp.) of that arc.

For an *arc*, there are some *line* attributes, which define the *style*, the *colour*, and the *width* in which it is displayed.

The *dimension* of a *node* or *page* gives its size, again as a pair referring to its width (x) and height (y). Depending on the ratio of height and width, a *place* is displayed as an ellipse rather than a circle. A *transition* is displayed as a rectangle of the corresponding size. If the dimension of an element is missing, each tool is free to use its own default value for the dimensions.

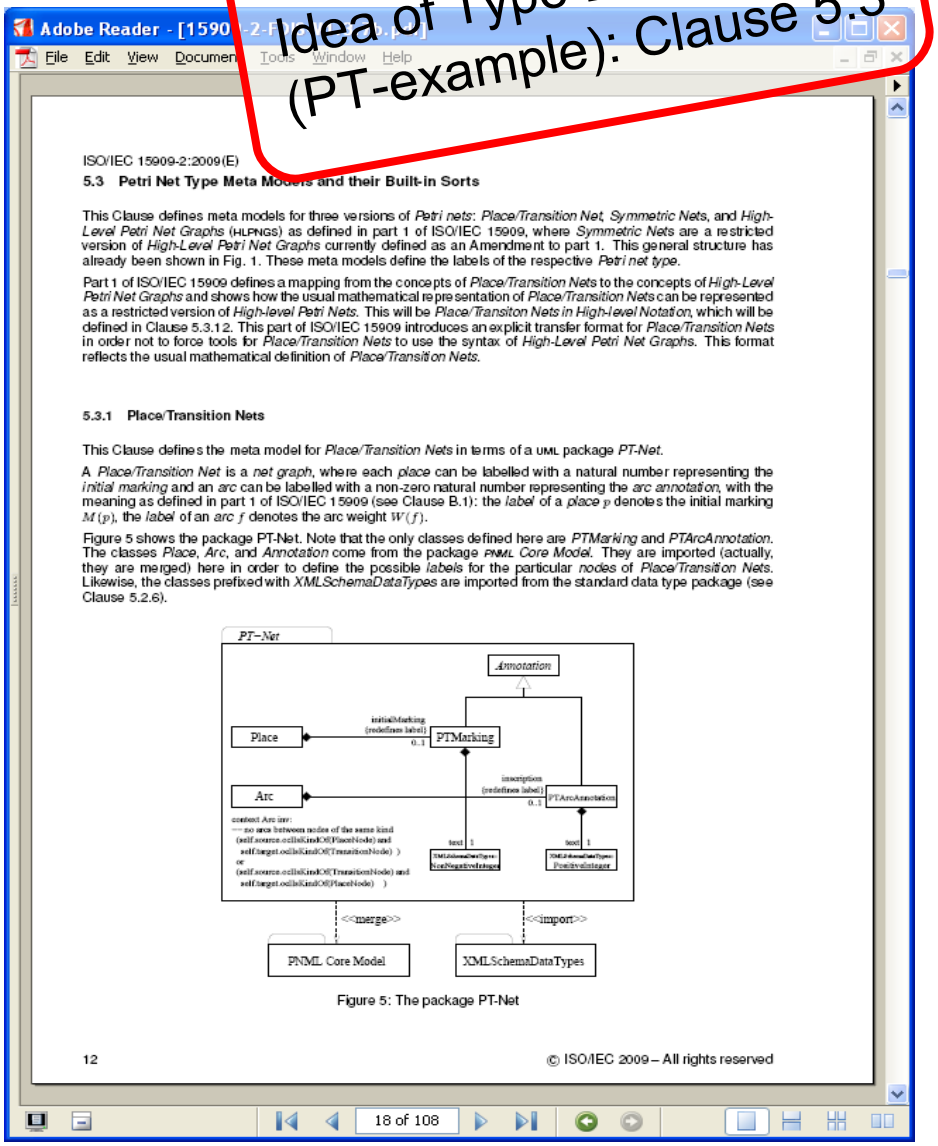
The two elements *fill* and *line* define the interior and outline colours of the corresponding element. The value assigned to a *color* attribute must be a RGB value or a predefined colour as defined by CSS2. When the attribute

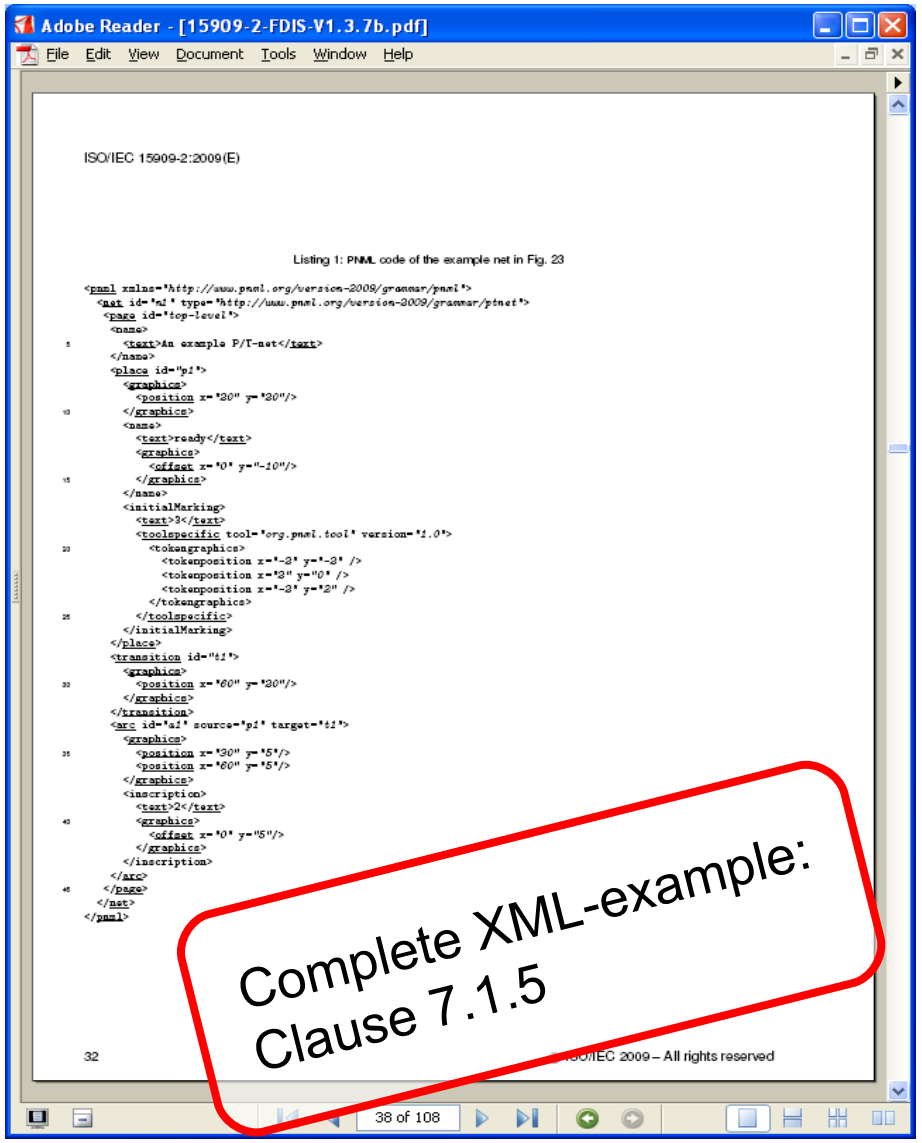
© ISO/IEC 2009 – All rights reserved

16 of 108

Place/Transition-Nets

Idea of Type Definition (PT-example): Clause 5.3





Complete XML-example: Clause 7.1.5

XML-Syntax

Mapping Core Model Concepts: Clause 7.1

Adobe Reader - [15909-2-FDIS-V1.3.7b.pdf]

File Edit View Document Tools Window Help

Table 4: Translation of the PNML Core Model into PNML elements

Class	XML element	XML Attributes
PetriNetDoc	<pnml>	xmlns: anyURI (http://www.pnml.org/version-2009/grammar/pnml)
PetriNet	<net>	id: ID type: anyURI
Place	<place>	id: ID
Transition	<transition>	id: ID
Arc	<arc>	id: ID source: IDRef (Node) target: IDRef (Node)
Page	<page>	id: ID
RefPlace	<referencePlace>	id: ID ref: IDRef (Place or RefPlace)
RefTrans	<referenceTransition>	id: ID ref: IDRef (Transition or RefTrans)
ToolInfo	<toolspecific>	tool: string version: string
Graphics	<graphics>	
Name	<name>	

Table 5: Possible child elements of the <graphics> element

Parent element class	Sub-elements of <graphics>
Node, Page	<position> <dimension> <fill> <line>
Arc	<position> (zero or more) <line>
Annotation	<offset> <fill> <line> <font>

In general PNML *Documents*, any XML element that is not defined in the PNML *Core Model* (i.e. not occurring in Table 4) is considered as a *label* of the PNML *element* in which it occurs. For example, an <initialMarking> element could be a *label* of a *place*, which represents its initial marking. Likewise <name> represent the *name*[4] of an *object*, and <inscription> an arc annotation.

A legal element for a *label* must contain at least one of the two following elements, which represents the actual value of the *label*: a <text> element represents the value of the *label* as a simple string; the <structure> element can be used for representing the value as an abstract syntax tree in XML.

An optional PNML <graphics> element defines its graphical appearance; and optional PNML <toolspecific> elements may add tool specific information to the label. Note that this part of ISO/IEC 15909 does not mandate the inner structure of <toolspecific> elements; every tool is free to structure its information inside that element at its discretion.

7.1.3 Graphics

All PNML *elements* and all *labels* may include graphical information. The internal structure of the PNML <graphics> element, i.e. the legal XML children, depends on the element in which the graphics element occurs. Table 5 shows the elements which may occur within the <graphics> element (as defined by the UML model in Fig. 3).

[4]Note that this, actually, is the only *label* that is explicitly defined in the *PNML Core Model*.

© ISO/IEC 2009 – All rights reserved

36 of 108

The <position> element defines an absolute position for *nodes* and *pages*, whereas the <offset> element defines a relative position for *annotation*. The name of the element comes from the corresponding role in the UML diagram, and the possible attributes are derived from the attributes of the corresponding class in the UML diagram.

Table 6 explicitly list the attributes for each graphical element defined in Table 5 (cf. Fig. 3 and Table 1). The domain of the attributes refers to the data types of either XML Schema, or Cascading Stylesheets 2 (css2), or is given by an explicit enumeration of the legal values.

Table 6: PNML graphical elements

XML element	Attribute	Domain
<position>	x	decimal
	y	decimal
<offset>	x	decimal
	y	decimal
<dimension>	x	nonNegativeDecimal
	y	nonNegativeDecimal
<fill>	color	CSS2-color
	image	anyURI
	gradient-color	CSS2-color
	gradient-rotation	{vertical, horizontal, diagonal}
<line>	shape	{line, curve}
	color	CSS2-color
	width	nonNegativeDecimal
	style	{solid, dash, dot}
<font>	family	CSS2-font-family
	style	CSS2-font-style
	weight	CSS2-font-weight
	size	CSS2-font-size
	decoration	{underline, overline, line-through}
	align	{left, center, right}
	rotation	decimal

The meaning of these elements and attributes is defined in Clause 5.2.4.

7.1.4 Mapping of XMLSchemaDataTypes concepts

The concepts from the package *XMLSchemaDataTypes* are mapped to XML syntax in the following way. The *String* objects are mapped to XML PCDATA, i.e. there will be a PCDATA section within the element which contains the *String*. This, basically, corresponds to any printable text.

Likewise, *Integers*, *NonNegativeIntegers* and *PositiveIntegers* are mapped to the XMLSchema syntax constructs *integer*, *nonNegativeInteger*, and *positiveInteger*, respectively.

7.1.5 Example

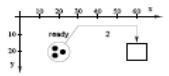


Figure 23: A simple *Place/Transition Net*

In order to illustrate the structure of a PNML *Document*, there is a simple example PNML *Document* representing the *Petri net* shown in Fig. 23, which actually is a *Place/Transition Net*. Listing 1 shows the corresponding PNML

© ISO/IEC 2009 – All rights reserved

37 of 108

High-level nets

HLPNG Core Structure:
Clause 5.3.2

Details of the built-in types:
Clause 5.3.3 – 5.3.9

Adobe Reader - [15909-2-FDIS-V1.3.7b.pdf]

File Edit View Document Tools Window Help

For constructing such *terms*, one can use built-in *operators* and *sorts*, and user-defined *variables*, which are defined in a *variable declaration*. The *variable declarations* are *annotations* of the *net* or a *page*. Moreover, a *transition* can have a *condition*, which is a *term* of *sort* boolean and imposes additional conditions on the situations in which a transition can fire. These concepts and their semantics are precisely defined in part 1 of ISO/IEC 15909. The only difference is that, in part 2, there is a fixed relation between *sorts* and *types* and between *operators* and *functions*.

The UML diagram with package *Terms*, which defines all these concepts, is shown in Fig. 7: It defines the concepts of *sorts*, *operators*, *declarations*, and *terms*, and how *terms* are constructed from *variables* and *operators*.

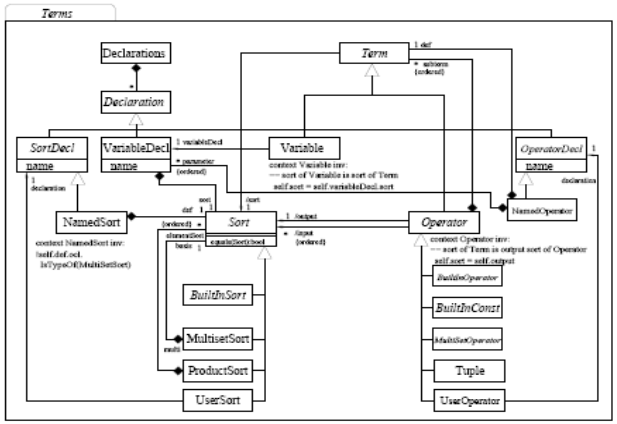


Figure 7: The meta model for *Terms*

For each *variable declaration* there is a corresponding *sort*. A *sort* can be a *built-in sort*, a *multiset sort* over some *basis sort*, a *product sort* over some *sorts*, or a *sort* which is given in a *user declaration*. In the core structure, the only possible *sort declaration* of a user is by constructing a new *sort* from existing ones and by giving them a new name. From these, the user can define new ones. Note that cyclic references in user defined *sorts* are not allowed. In addition to these user defined sorts, called *named sorts*, there will be *arbitrary sorts*. Since these are not allowed in *Symmetric Nets* these will be defined only in Clause 5.3.11.

An *operator* can be a *built-in constant* or *built-in operator*, a *multiset operator* which among others can construct a *multiset* from an enumeration of its elements, or a *tuple* operator. Each *operator* has a sequence of *sorts* as its *input sorts*, and exactly one *output sort*, which defines its signature. As for *sorts*, the user can define own *operators*. Here, it is only possible to define an abbreviation, which is called a *named operator*: It can use a *term*, which is built from existing *operators* and parameter variables, for defining a new *operator*. As for *sorts*, cycles (recursion) in these definitions are not allowed. As for sorts, there will be arbitrary operator declarations for *High-level Petri Net Graphs*, but not for *Symmetric Nets*. Therefore, these concepts will be defined in Clause 5.3.11.

From the built-in *operators* and the user-defined *operators* and *variables*, *terms* can be constructed in the usual way. The *sort* of a *term* is the *sort* of the *variable* or the *output sort* of the *operator*. Therefore, *sort* is indicated as a derived association; its definition is expressed by an OCL expression in the UML meta model. Note that the *input*

© ISO/IEC 2009 – All rights reserved

Adobe Reader - [15909-2-FDIS-V1.3.7b.pdf]

File Edit View Document Tools Window Help

output sort of the *operator* are also represented as derived associations because, in some situations, they need not be given explicitly since they can be derived from the type of the *operator*.

Figure 8 shows the package *High-Level Core Structure*, which defines all the *annotations* for both, *Symmetric Nets* and *High-Level Petri Net Graphs*. Note that, since the classes for built-in *sorts* and *operators* are abstract, we do not have any built-in *sorts* and *operators* yet. These will be defined in Clauses 5.3.3 to 5.3.9.

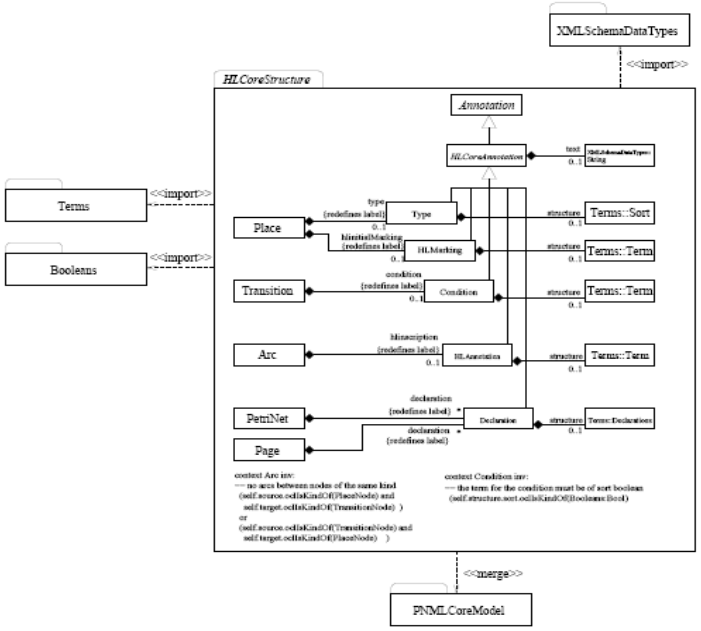


Figure 8: The package *High-Level Core Structure*

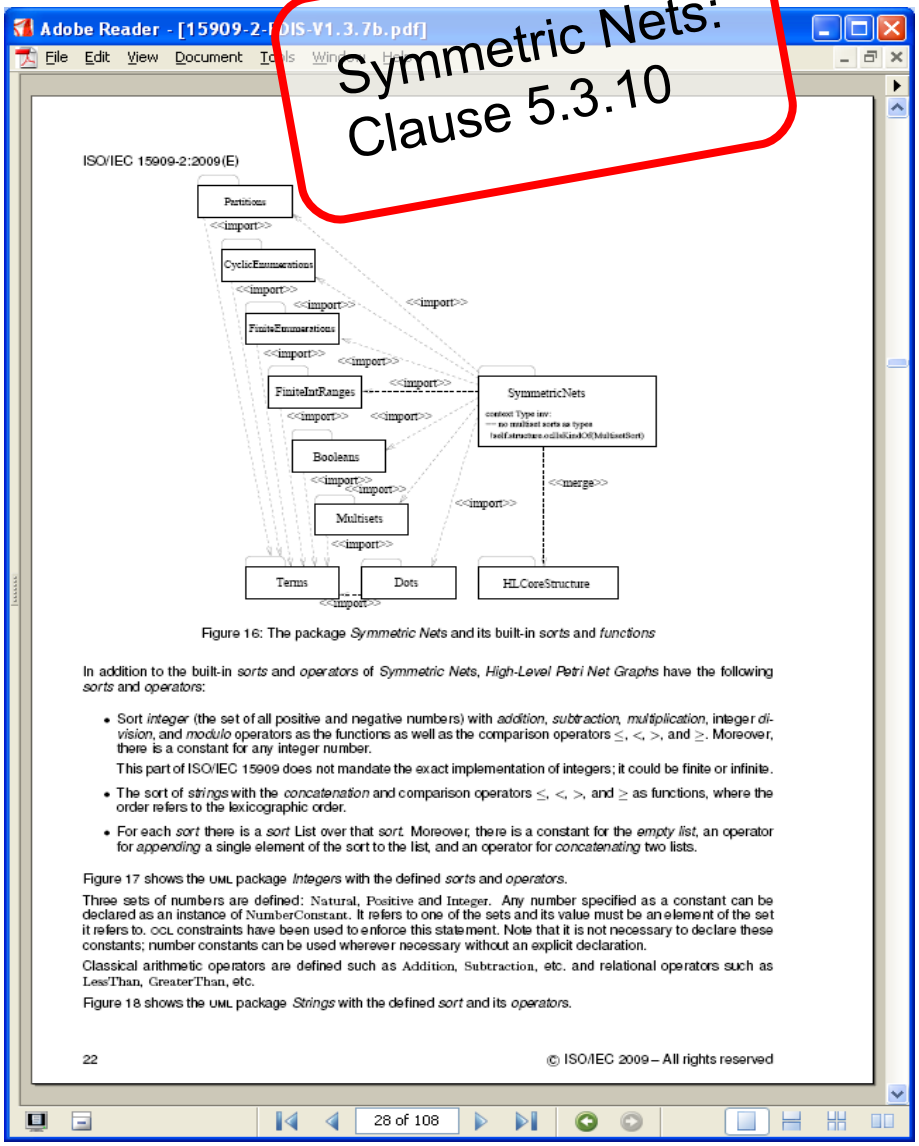
In addition to the *annotations* defined above, the package *High-Level Core Structure* requires that *arcs* must not connect two *places* and must not connect two *transitions*.

Note that this model defines an abstract syntax (*structure*) for all these concepts only. In order to allow tools to store some concrete text, all *annotations* of *High-level Nets* may also consist of text, which should be the same

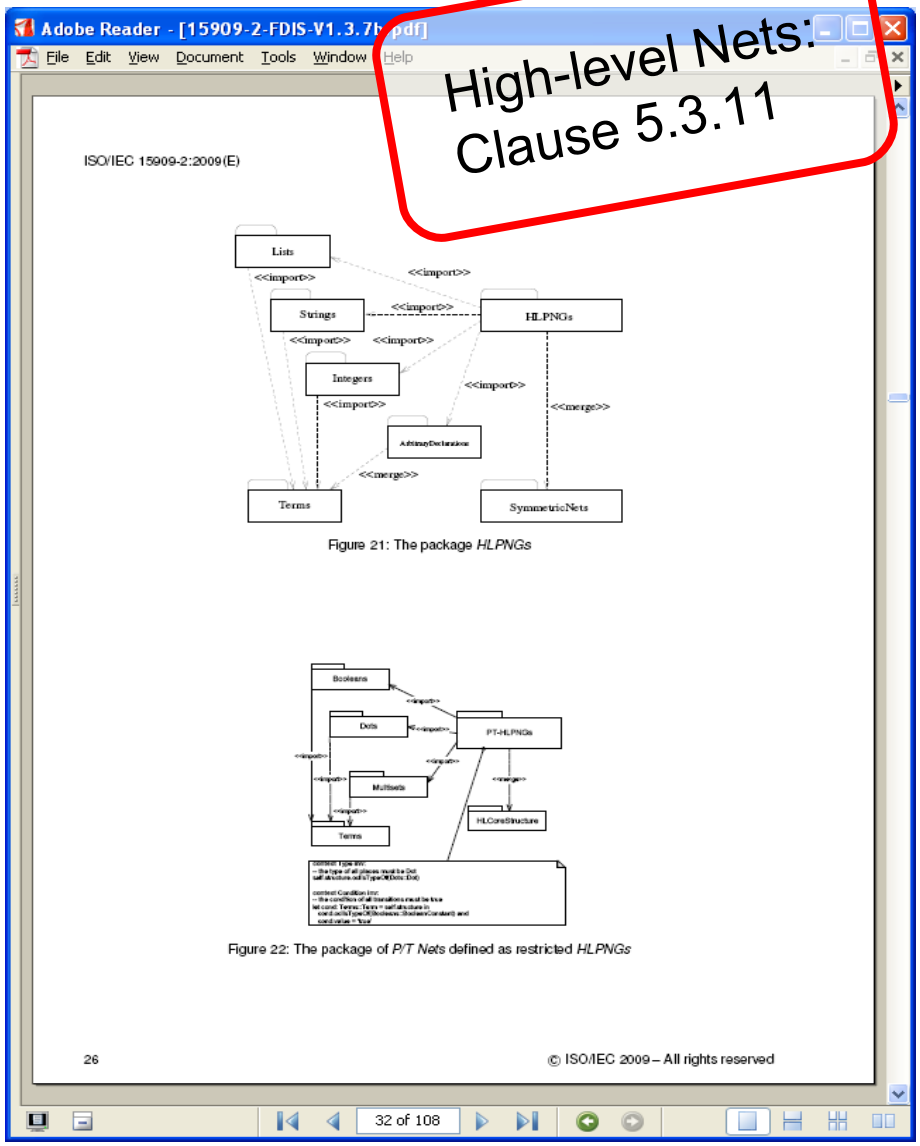
© ISO/IEC 2009 – All rights reserved

SN and HLPNGS

Symmetric Nets:
Clause 5.3.10



High-level Nets:
Clause 5.3.11



Status

Presented Part:

- ISO 15909-2 (1.3.7), FDIS currently
- PNML Core Model, P/T-Nets, HLPNGs und Symmetric Nets
- Tool support:
  Many tools supporting (variants of) PNML
  PNML Framework (second part)

Open issues:

- Explicit Petri net type definitions
- Modules
- …

Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
- Extensions and Current & Future Work
  - Modular PNML
  - Type and feature definitions
  - …
- Conclusions

Modular PNML

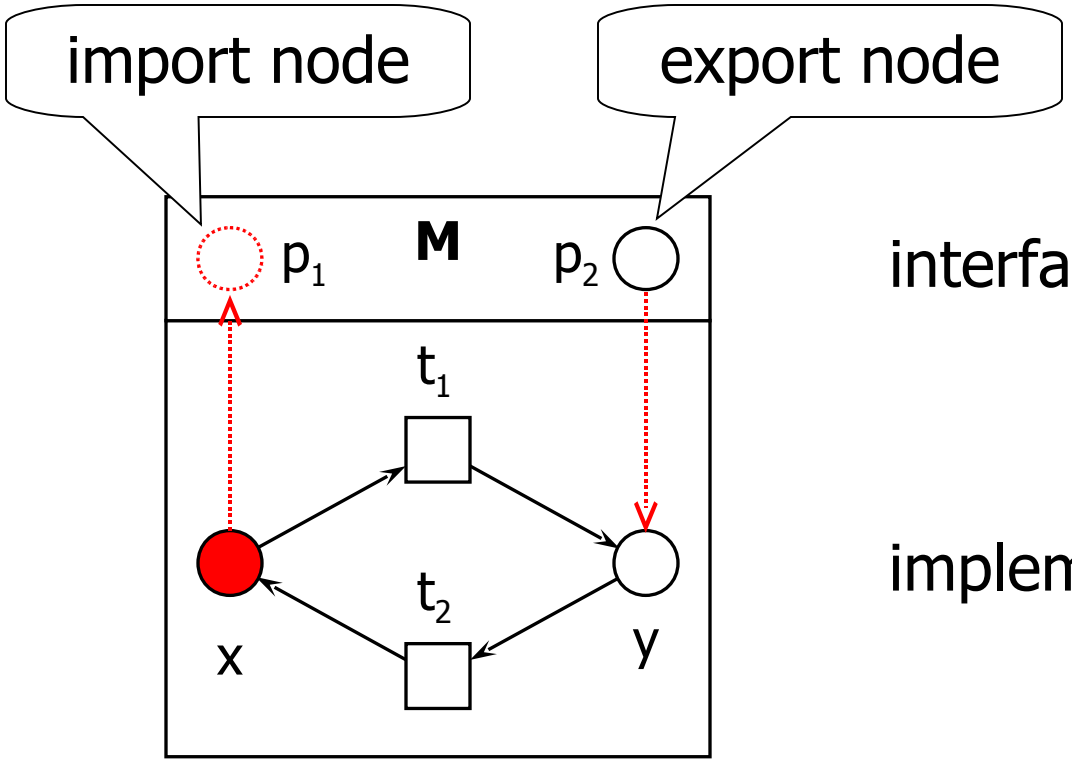
Goals:

- structuring (of large nets)
- re-use

Concepts (1):

- **module** definitions
- module **instances**

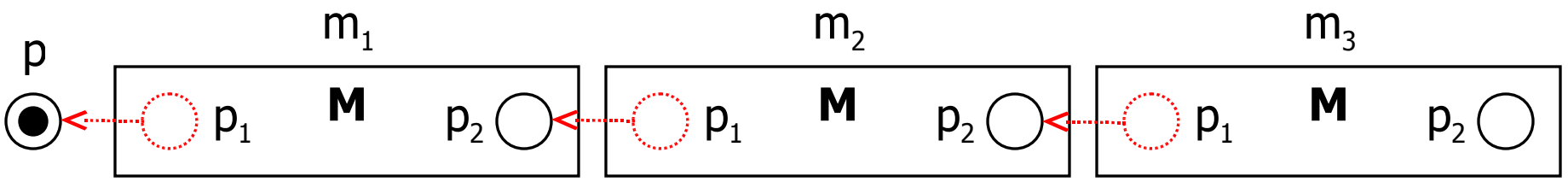
Module Definition



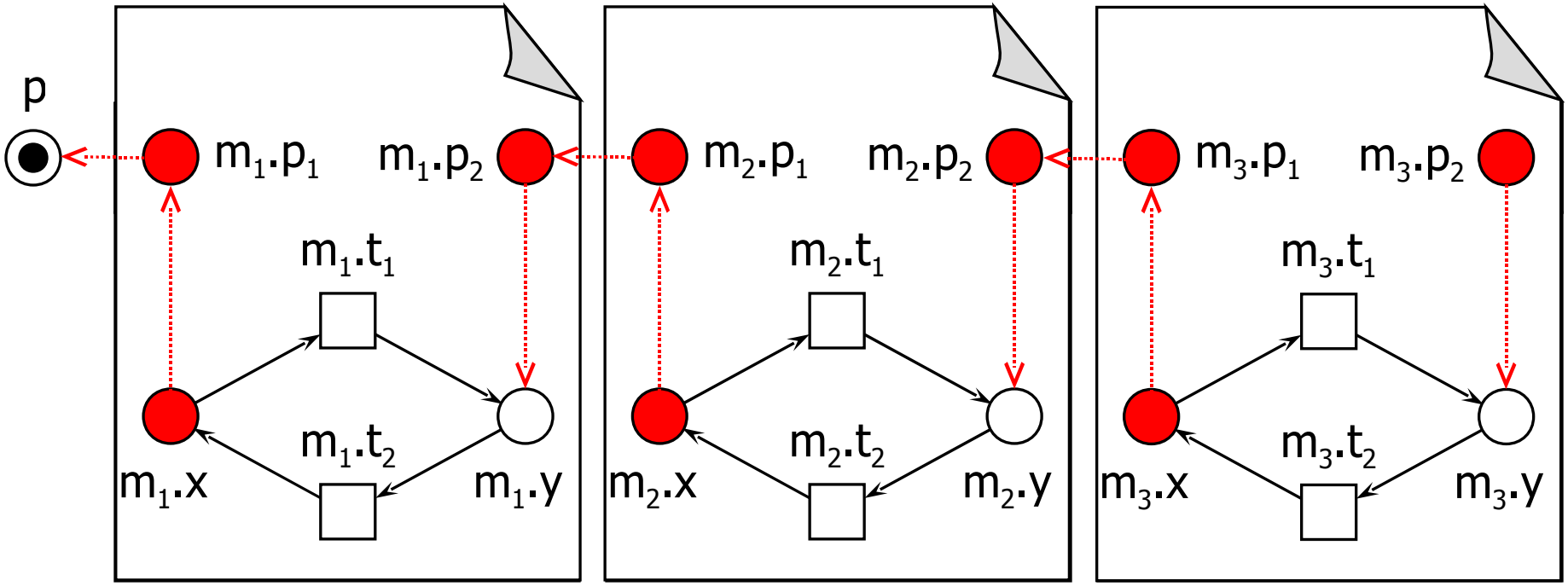
interface

implementation

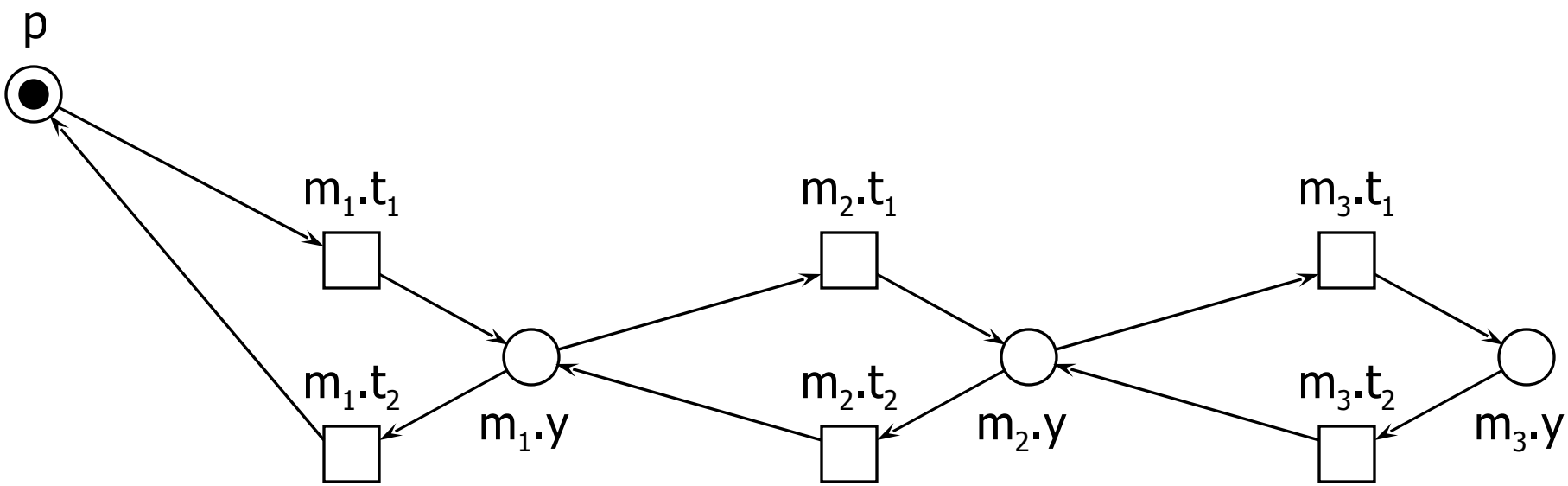
Module Instances



Semantics: a. „Inlining“



Semantics: b. „Flattening“



Modular PNML

Goal:

- sharing “labels” (symbols)

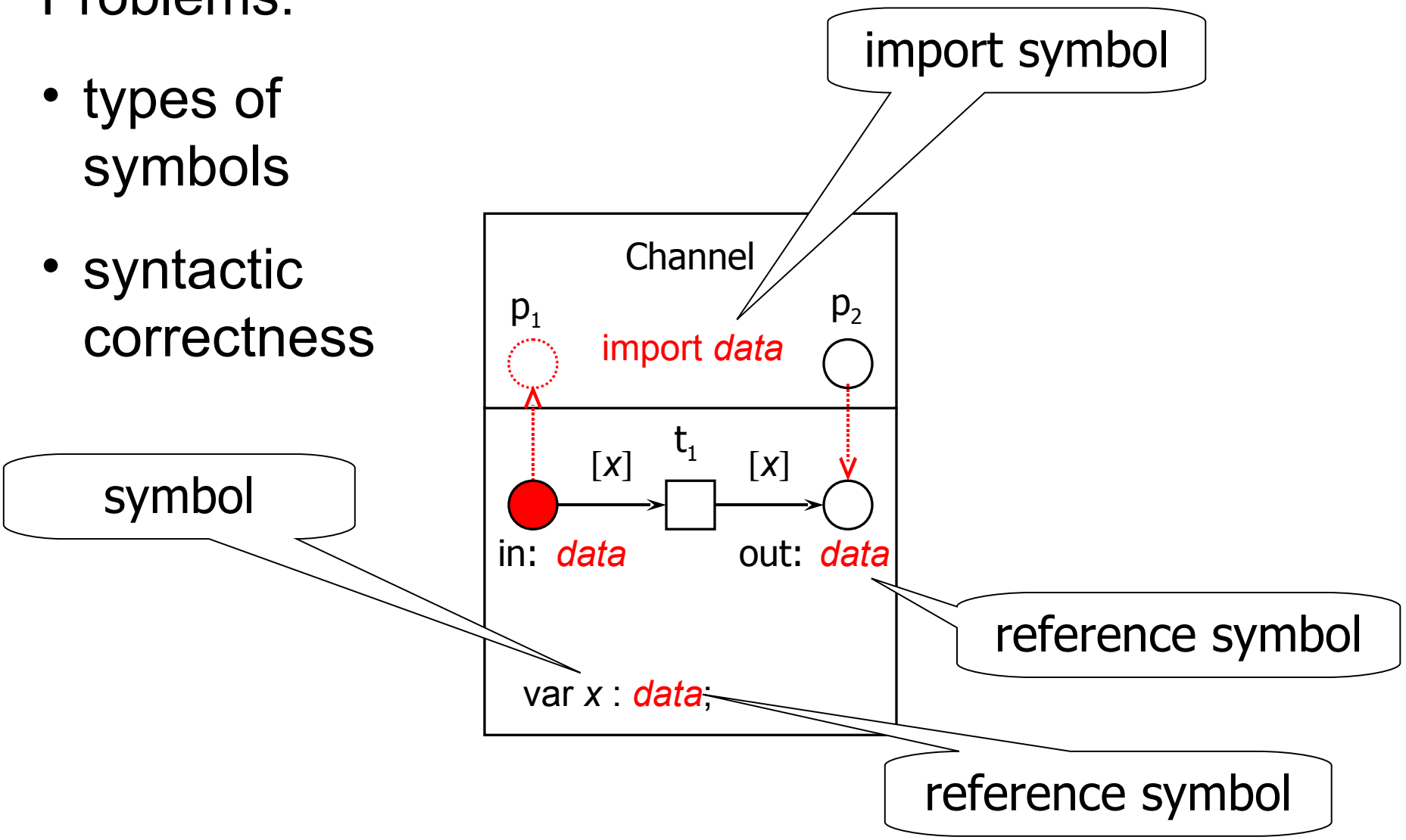
Concepts (2):

- symbols
- import- and export symbols

Example: Symbols

Problems:

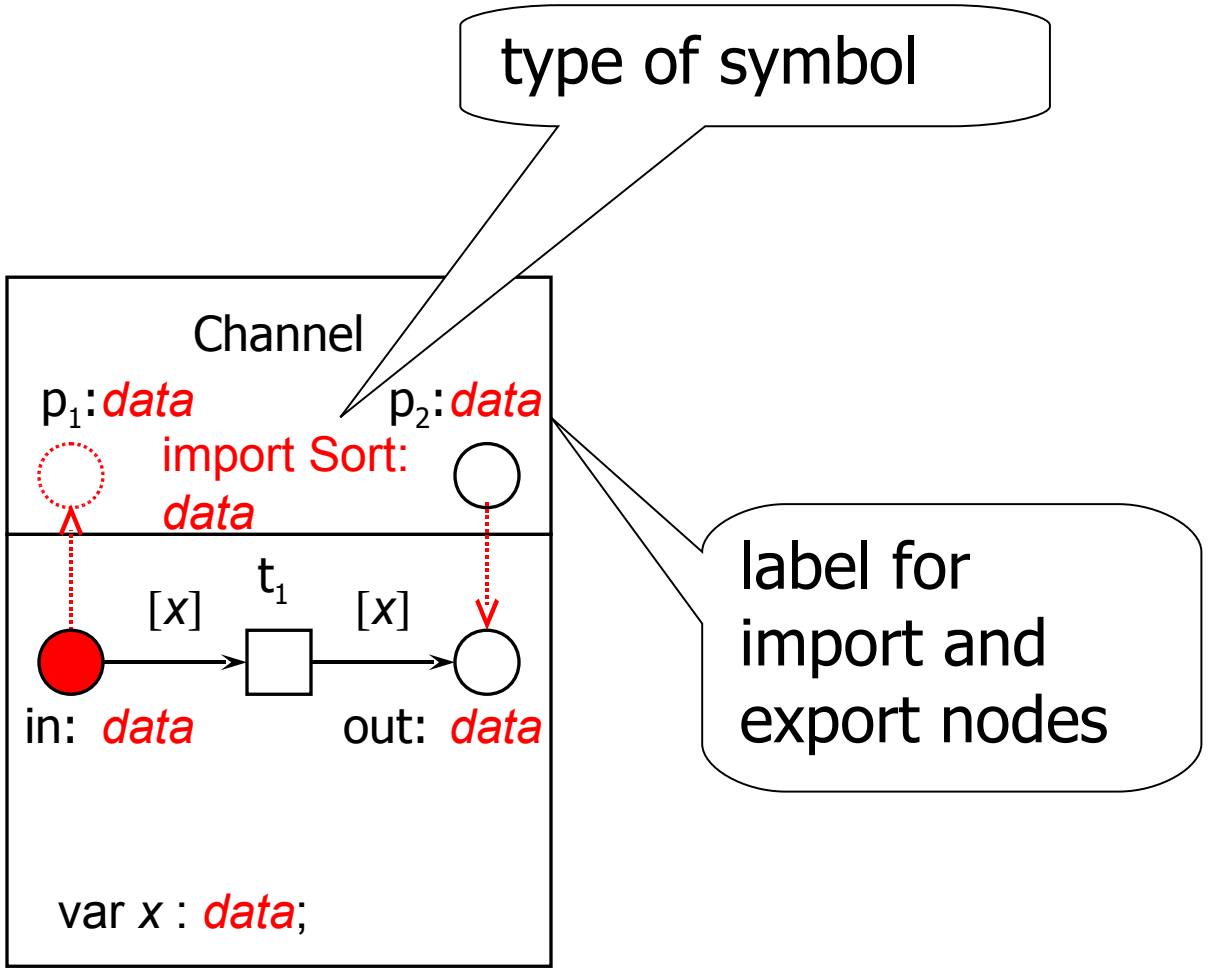
- types of symbols
- syntactic correctness



Concept 2 (extended)

Problems:

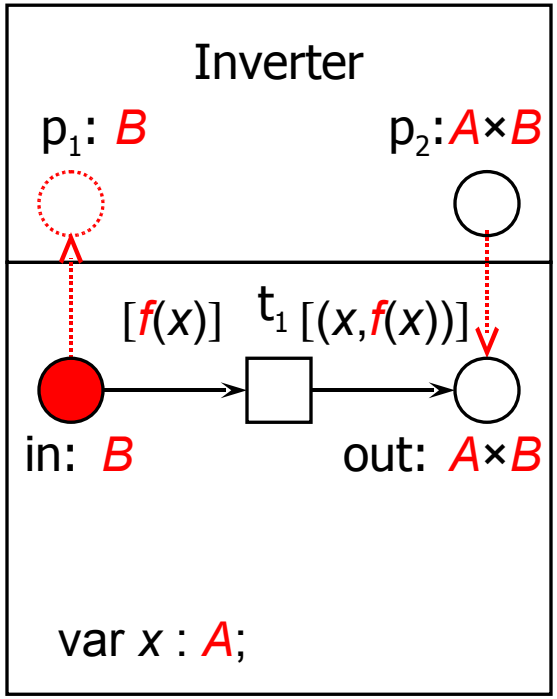
- Label of reference nodes have a meaning now
- How does PNML "know", which symbols are there?
- More complex symbols



Concept 2 (extended)

Problems:

- How does PNML "know", which symbols are there?
- How does PNML "know", which information must be provided for "complex symbols"?

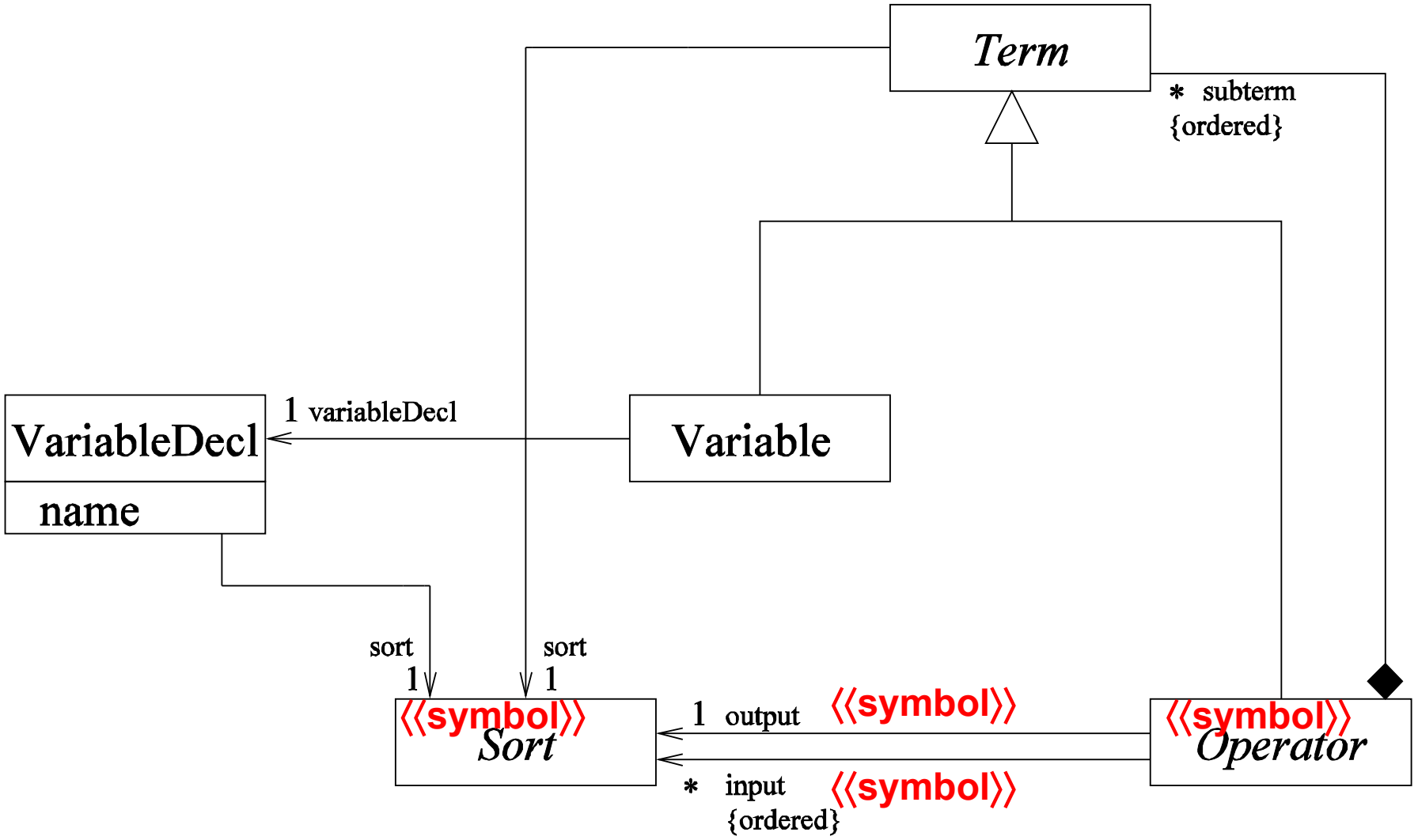


import Sort: *A*

import Sort: *B*

import Operator: *f*
input Sort: *A*
output Sort: *B*

Concept 2: Solution



Modular PNML

- Is that it?

- We presume so!
- Anybody needing more, speak up?
- See talk at conference tomorrow!

- Do we need export symbols?

Don't know!

Do no harm!

- How does the XML look like?

To be discussed

Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
- Extensions and Current & Future Work
- Conclusions

Summary and outlook

- PNML is an XML-based transfer format for “all kinds” of Petri nets
- Standard ISO/IEC 15909-2 (Focus: High-level nets)
- ISO 15909-3++
  - API
  - Petri Net Type Definition Interface
  - Petri Net Type Definitions
  - Modularity
  - More features

Credits

Jonathan Billington
Søren Christensen
Jörg Desel
Erik Fischer
Giuliana Franceschinis
Kees van Hee
Lom Hillah
Nisse Husberg
Matthias Jügel
Albert Koelmans
Fabrice Kordon
Olaf Kummer
Kjeld Høyer Mortensen
Laure Petrucci
Renier Post
Wolfgang Reisig
Stefan Roch
Karsten Schmidt (Wolf)
Christian Stehno
Nicolas Trèves
Kimmo Varpaaniemi
Michael Weber
Lisa Wells
...

http://www.pnml.org/

Discussion

- Any question?
- Any problems, criticism, or proposals?

- Any new ideas?
- Any features or Petri net types missing?

→ Just join in the work of ISO/IEC JTC1 SC7 WG19

Petri Nets 2009

The Petri Net Markup Language

theory and practice

Lom Messan Hillah
Université Pierre et Marie Curie

Outline

- Motivations
- PNML Framework: how to use it?
- Application examples (Coloane, Validation, Dot)
- How is it built? (MDE)

Outline

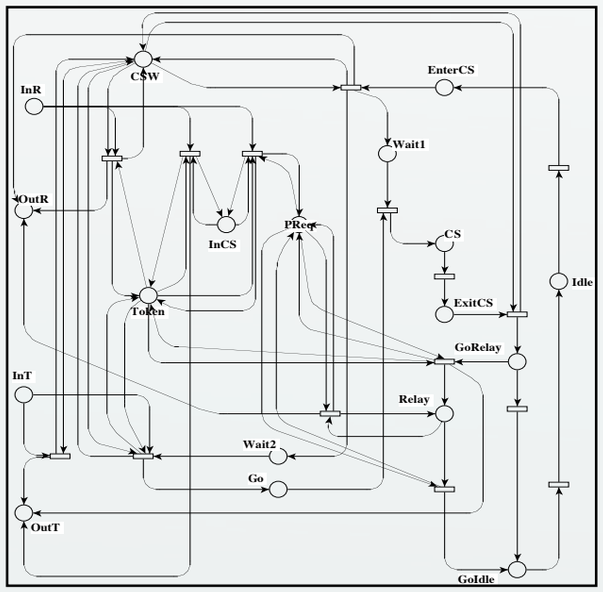
- Limitations
- Ideas for improvement
- Conclusion
- Resources

Exchanging PN models...

- PNML is about exchanging Petri net models, not XML

PN designer

Import



Export

PN designer

A Petri net model

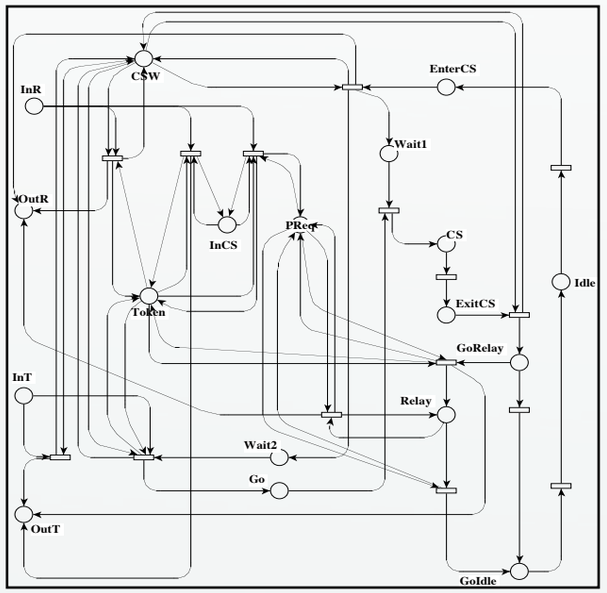
Exchanging PN models...

- The XML syntax is irrelevant to design
- Tools must (automatically) deal with it
- Keep compliance with the standard
- Tool-specific tag for non-standard information
- Best effort strategy otherwise

Exchanging PN models...

PN designer

Import



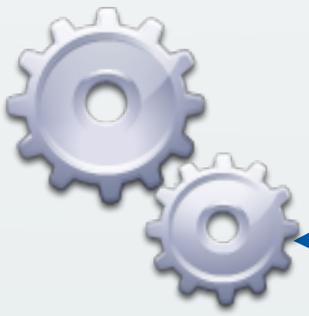
Export

PN designer

Actually, what are their respective tools doing ?

FETCH

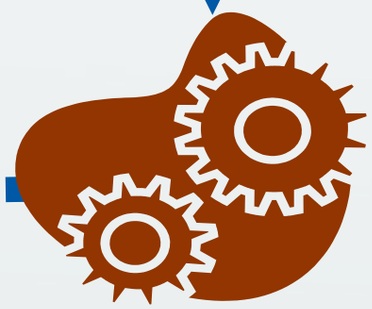
CREATE



LOAD

```
<pnml xmlns=".....">
 <net type="http://www"
  <name>
    dining philosophers
  </name>
  <page id="page1">
   <place id="p1">
```

SAVE



PNML Document

Outline

Motivations

PNML learning difficulties

- x Developers are not yet familiar with the standard
- x Conceptual part is the one to fully comprehend
- x The core semantics do not lie in the XML
- x The standard is not freely available

We need, at least, an entry point

Easing the access to PNML

- ☑ Transparent, easy way to handle PNML documents
- ☑ Help developers concentrate on the core of their applications, not PNML
- ☑ Keeping up-to-date and compliant
- ☑ Keep the door open for future extensions
- ⚙ Reference implementations should make it work

Outline

Motivations

PNML Framework

Motivations

PNML Framework

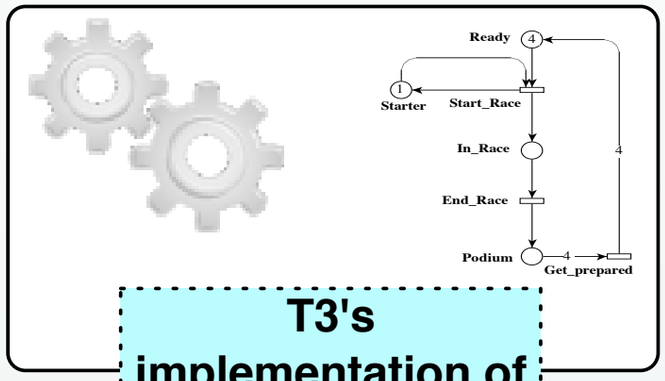
- Aims at being a reference implementation of PNML
- It is first intended to be used as a library by tools

➡ Easy to use API to handle PNML documents

- Two use cases of a PN tool for handling PNML:
  - import Petri nets from PNML documents
  - export Petri nets into PNML documents

PNML Framework at work

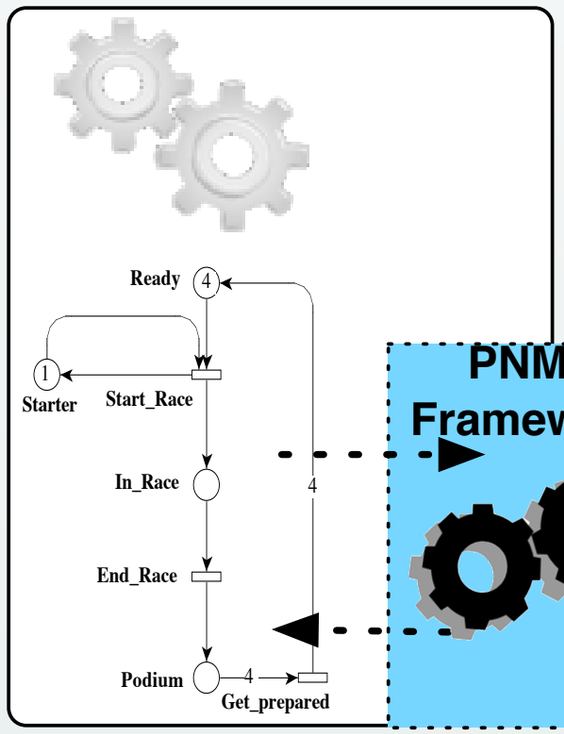
Petri net tool T3



T3's implementation of PNML

export **import**

Petri net tool T1



PNML Framework

export **import**

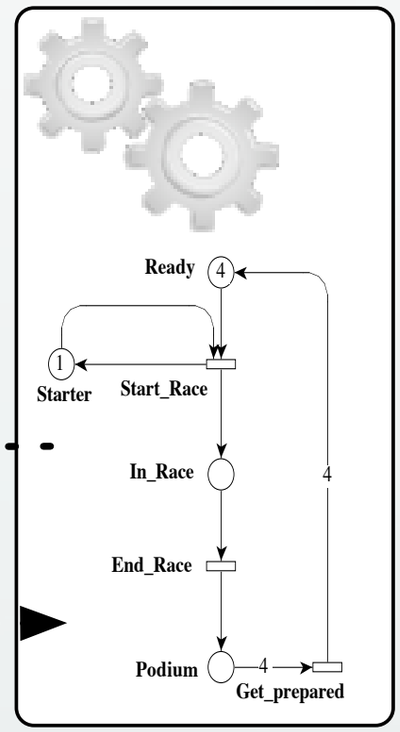
Best effort PNML handling

PNML document

PNML Framework

export **import**

Petri net tool T2



PNML Framework integrated in an existing tool

PNML Framework integrated in a new independent tool

Integrating PNML Framework

Petri nets tool developer

implements

Developer's PNML handler

Application Driver

PNML Framework

Create models

PNML repository

4

Fetch

3

Save models

Fetch and check

Petri net models in proprietary format

1

Load

Developer's parser

Populate repository

2

Proprietary model repository

5

PNML exporter

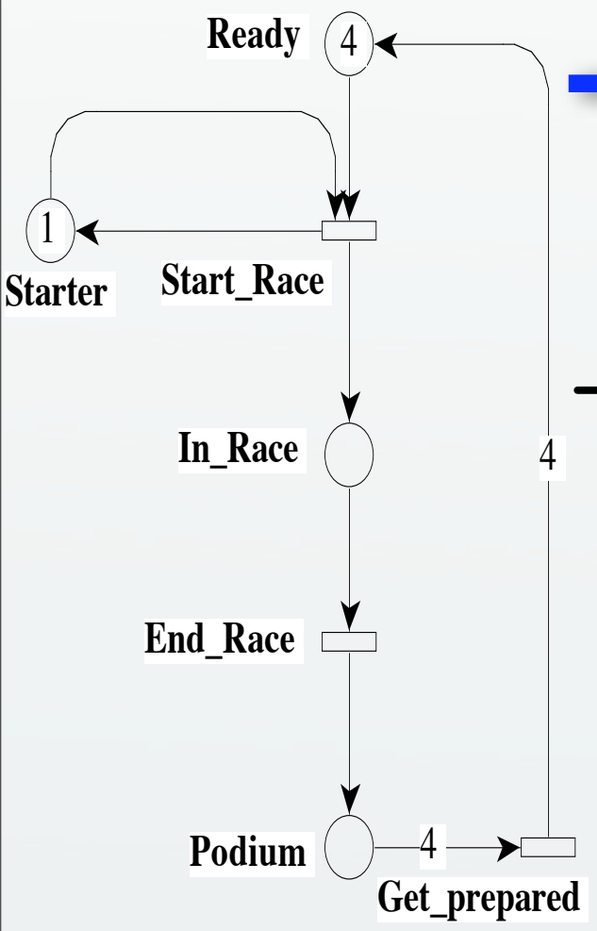
Save

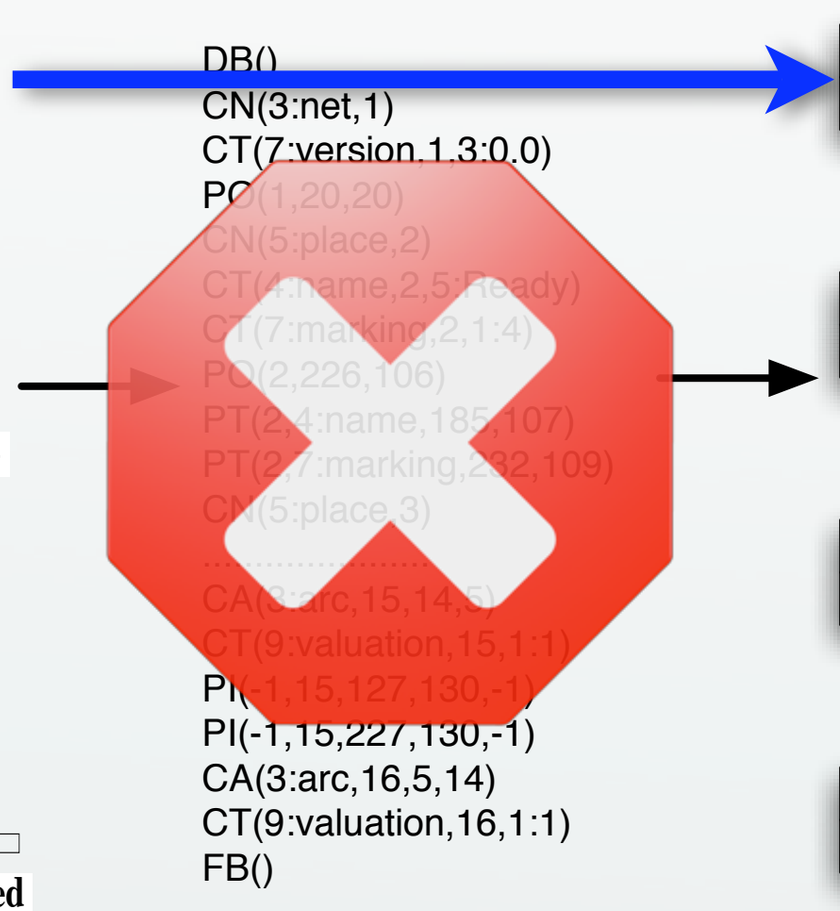
6

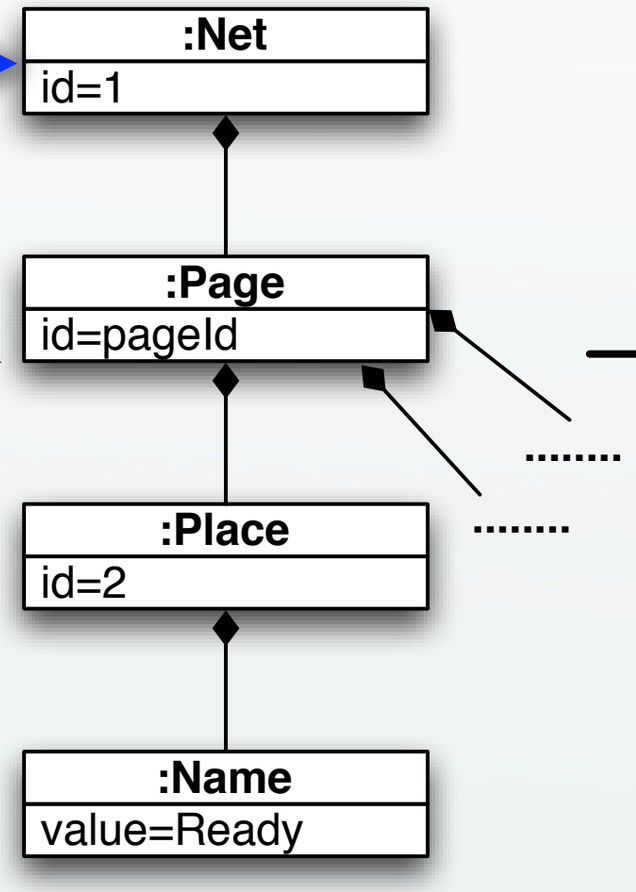
Petri net models in PNML

Developer's part

Transformation chain








```
<pnml xmlns="...">
 <net id="1">
  ..............................
  <page id="pageId">
   ..........................
   <place id="2">
    <name>
     <text>Ready</text>
      ..............................
     </name>
     ..............................
     <initialMarking>
      <text>4</text>
     </initialMarking
      ......................
      ......................
</pnml>
```

A **B** **C** **D**

Developer's tool part

PNML Framework part

Outline

Motivations

PNML Framework

Application examples

Application examples (Short demo)

- Coloane (cross-platform Petri net editor)
- PNML validation
- PNML to dot
- PNML to Coq

Outline

Motivations

Application examples

PNML Framework

How is it built?

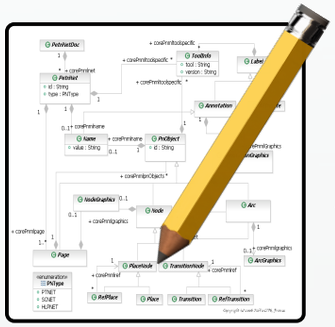
How is it built?

- Model-Driven Engineering principles:
- First model your domain-specific language (DSL)
- Customize code generation
- Then generate code, customize again, round-trip, ...
- Advantages: high-level, less error-prone, target language (relatively) independent, sync. model-code, maintainable, extensible

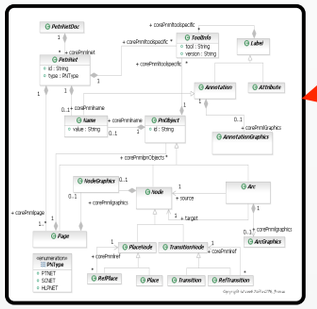
How is it built?

metamodels for Petri net types

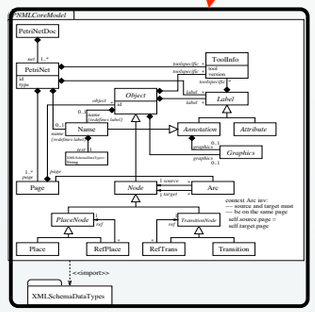
annoted model for code generation



Petri net type T1 genmodel (EMF)



Petri net type T1 Ecore model (EMF)



Petri net type T1 Standard version

1

2

3

```
<%Eclassifier cl = ... %>
<%if (isInterface) {%>
/** Export to PNML */
public String toPNML();
<%}%>
<%.....%>
```

Code generation templates (configuration for PNML)

4

4

4

JET Code Generator

Code generation

5

```
/** Export to PNML */
public String toPNML();
.....
```

PNML Export API

```
/** Import from PNML */
public pnModel
fromPNML();
.....
```

PNML Import API

CREATE SAVE

LOAD FETCH

Outline

Motivations

Application examples

PNML Framework

Limitations / Improvement

How is it built?

Limitations

- Static code generation API, w.r.t. the metamodels
- Every metamodel update implies a code re-generation
- No native front-end, i.e., editor (but a lot of them out there + new initiatives)
- Java

Ideas for improvement

- Dynamic metamodel plug-in mechanism
- Grammar generation for new Petri net types
- New target languages

Outline

Motivations

Application examples

Conclusion / Resources

PNML Framework

How is it built?

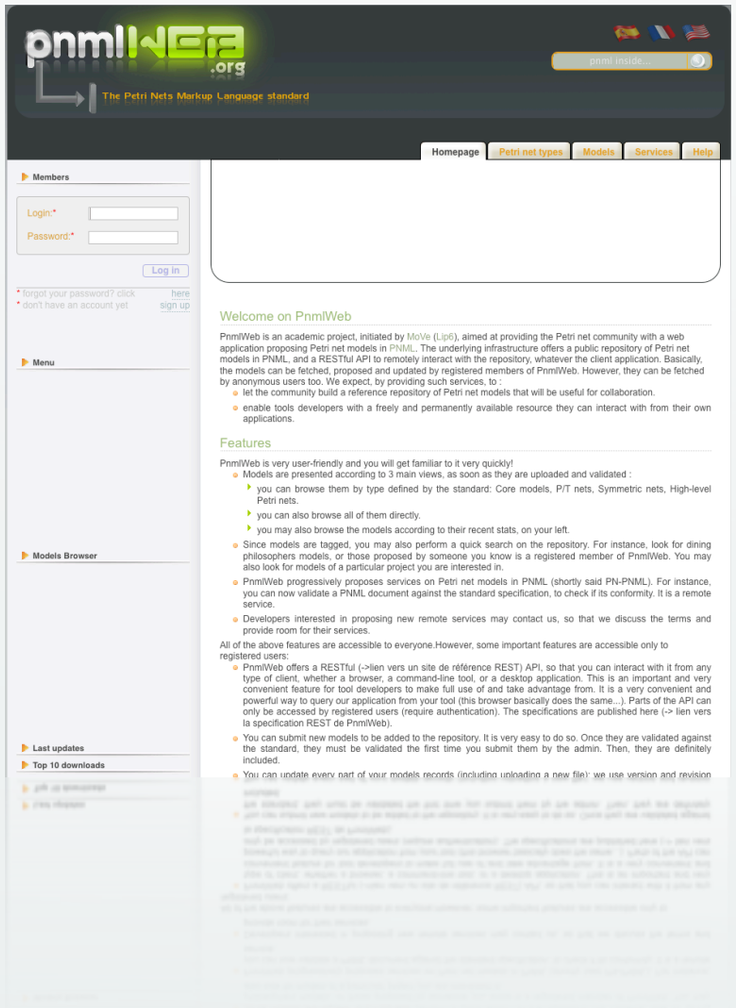
Limitations/ Improvement

PNML: theory and practice

- Interoperability of Petri net tools
- Designed for extension
- PNML Framework: a reference implementation for PN tools to use to handle PNML documents
- Easy to use, fast integration
- New applications are welcome

PNML-WEB

- Web-based repository for PN models in PNML
- Freely accessible: you can download or propose models
- Free and immediate PNML document validation
- RESTful: you can interact with it from your own application (URL-based invocations)



Webography

- http://www.pnml.org
- http://pnml.lip6.fr
- http://coloane.lip6.fr
- http://www.pnmlweb.org (Fall 2009)