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
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
  - …
Isn’t XML just sooooo boring?

That’s why the focus here is on concepts.

Agreed!
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 15909-2 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.
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.
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

```xml
<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

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

```xml
<place id="pl">
    <name>
        <text>i</text>
    </name>
    <initialMarking>
        <text>1</text>
    </initialMarking>
</place>
```
Basic Idea

„All kinds“ of Petri nets can be represented by

- places
- transitions, and
- arcs

along with some

- labels

PNML Core Model

«merge»

Petri Net Type
Definition
Outline

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

- **PetriNet**
  - **Node**
    - **Place**
    - **Transition**
  - **Arc**
    - **Label**

The diagram illustrates the components of a Petri net model within the PNML Core Model. The objects are connected with arrows indicating relationships such as source and target. The label is connected to the arc, indicating the action. The numbers and arrows in the example Petri net show the flow and states transition.
Core Model (overview)

PetriNet

Object

Node

Arc

Place

Transition

Label

PetriNet

Object

Node

Arc

Place

Transition

Label

PetriNet

Object

Node

Arc

Place

Transition

Label
Core Model (overview)

PetriNet

Object

Node

Place

Transition

Arc

Label

Annotation

Attribute

PNML: Theory
Type Definition: PT-Net

Place

Arc

initialMarking

PTMarking

{redefines label}

Annotation

{redefines label}

PTInscription

inscription

text

XML_Schema::NonNegativeInteger

XML_Schema::PositiveInteger

PNML: Theory
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) )
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
<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"/>
  </net>
</pnml>
Labels in XML

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

PNML Core Model

PT-Net

XML Schema::NonNegativeInteger

PTMarking

text

Node

Name

text

initialMarking

Place

PNML: Theory
Outline

- Introduction and Motivation
- Basic concepts
- Mapping to XML
- More details
  - Tool-specific 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**

**Context Arc inv:**
- source and target must be on the same page
  - self.source.page = self.target.page
<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 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

PNML: Theory
Pages and reference nodes

request1

critical1

ref1

idle1

request2

critical2

ref2

idle2

semaphor
Pages and references

**Object**

context Arc inv:
- self.source.page = self.target.page

**Node**

**Place**

**Transition**

**Page**

**RefPlace**

**RefTrans**

**RefNode**

**PlaceNode**

**Transition Node**

- source
- target

context Arc inv:
- self.source.page = self.target.page
Always at least one page!

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

X

negotiating: A

envelopes: A×A

agreed: A

mailbox: A×A
Sorts, operators, terms

PNML: Theory
context Arc inv:
  no arcs between nodes of the same kind
  (self.sourceoclIsKindOf(PlaceNode) and
   self.targetoclIsKindOf(TransitionNode) )
  or
  (self.sourceoclIsKindOf(TransitionNode) and
   self.targetoclIsKindOf(PlaceNode) )

context Condition inv:
  the term for the condition must be of sort boolean
  (self.structure.sortoclIsKindOf(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

PT-Nets

Symmetric Nets

HLPNGs

PNML Core

«merge»

«merge»

«merge»
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**
PNML: Theory

Place/Transition-Nets

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

Complete XML-example: Clause 7.1.5
The "position" element defines an absolute position for nodes and pages, whereas the "refposition" element defines a relative position for annotation. The name of the element comes from the corresponding node in the UML diagram, and the possible attributes are derived from the attributes of the corresponding class in the UML diagram.

Table 7 explicitly lists the attributes for each graphical element defined in Tables 5 or Fig. 3 and Table 6. The domain of the attributes refers to the data types of the root, xml, or SVG, or is given by an explicit enumeration of the legal values.

### 7.1.4 Mapping of XMLSchemaDataTypes concepts

The concepts from the package XMLSchemaDataTypes are mapped to new, syntax in the following way. The string objects are mapped to an XMLSchema atom element which contains the element. This is expressed in the XML, and corresponds to tree-structured text.

### 7.1.5 Example

![Diagram](image.png)
High-level nets

HLPNG Core Structure

Clause 5.3.2

Details of the built-in types:

Clause 5.3.3 – 5.3.9
Symmetric Nets: Clause 5.3.10

High-level Nets: Clause 5.3.11
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

import node

interface

export node

implementation
Module Instances

PNML: Theory
Semantics: a. „Inlining“
Semantics: b. „Flattening“
Goal:
- sharing “labels” (symbols)

Concepts (2):
- symbols
- import- and export symbols
Example: Symbols

Problems:

• types of symbols
• syntactic correctness

import symbol
symbol
reference symbol
reference symbol

var \( x : data \),

Channel

\( p_1 \)
\( p_2 \)

\([x]\)

\( t_1 \)

\([x]\)
in: data
out: data

import \( data \)
Concept 2 (extended)

Problems:

• Label of reference nodes have a meaning now

• How does PNML “know”, which symbols are there?

• More complex symbols

```plaintext
Channel

p₁: data
import Sort: data

var x: data;

p₂: data
out: data
```

label for import and export nodes

type of symbol
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

Inverter

\[ p_1: B \quad p_2: A \times B \]

\[ \text{in: } B \quad \text{out: } A \times B \]

\[ \text{var } x : A; \]

\[ [f(x)] \to t_1 [(x, f(x))] \to y \]
Concept 2: Solution

VariableDecl
   name

Sort
   sort

Term
   * subterm
   {ordered}

Operator
   * input
     {ordered}

Variable
   variableDecl

1 variableDecl

1 output

1 sort

1 sort
Modular PNML

- Is that it?
- Do we need export symbols?
- How does the XML look like?

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

Don’t know!
Do no harm!

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
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

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...

Actually, what are their respective tools doing?

Actually, what are their respective tools doing?

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

PNML Document
Outline

Motivations
PNML learning difficulties

- Developers are not yet familiar with the standard
- Conceptual part is the one to fully comprehend
- The core semantics do not lie in the XML
- 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
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 T1

Petri net tool T2

Petri net tool T3

Petri net tool T1

Petri net tool T2

Petri net tool T3

Best effort PNML handling

PNML Framework integrated in an existing tool

PNML Framework integrated in a new independent tool

PNML document

PNML Framework at work

T3's implementation of PNML

PNML Framework integrated in an existing tool

 PNML Framework integrated in a new independent tool

Ready 4

In_Race

Podium

Get_prepared

Start_Race

End_Race

Get_prepared

Starter

Petri net tool T3

Petri net tool T2

Petri net tool T1

PNML Framework

PNML document

export

import

export

import

export

import

export

import
Integrating PNML Framework

Petri nets tool developer

1. Load Petri net models in proprietary format

2. Populate repository with models using the developer's parser

3. Fetch models from repository

4. Create models using the PNML Framework

5. Save models and check them using the PNML repository

6. Save models in PNML format

Developer's part

Application Driver

PNML Framework

PNML repository

Developer's PNML handler

Implementing Petri nets tool developer

CN(5:place,151)
CT(4:name,165,Cpt2)
CN(1:transition,164)

Petri net models in proprietary format

CNML: Practice - PN'09
Transformation chain

A: 

1. Starter
2. Start_Race
3. Ready
4. In_Race
5. End_Race
6. Podium
7. Get_prepared

B: 

DB()
CN(3:net,1)
CT(7:version,1:3:0.0)
PO(1:20,20)
CN(5:place,2)
CT(4:name,2,5:Ready)
CT(7:marking,2,1:4)
PO(2:226,106)
PT(2:4:name,185,107)
PT(2:7:marking,282,109)
CN(5:place,3)
CA(3:arc,15,14,5)
CT(9:valuation,15,1:1)
PI(-1,15,127,130,-1)
PI(-1,15,227,130,-1)
CA(3:arc,16,5,14)
CT(9:valuation,16,1:1)
FB()

C: 

:Net
id=1

:Page
id=pageld

:Place
id=2

:Name
value=Ready

D: 

<pnml xmlns="...">
<net id="1">
...........................
<page id="pageld">
...........................
<place id="2">
<name>
<text>Ready</text>
...........................
</name>
...........................
</place>
...........................
</page>
...........................
</net>
...........................
</pnml>

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

PNML Framework

How is it built?

Application examples
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?

1. Ecore model (EMF) - Petri net type T1
2. Annotated model for code generation
3. Code generation templates (configuration for PNML)
4. JET Code Generator
5. PNML Export API

- Petri net type T1
- Ecore model (EMF)
- PNML Export API
- PNML Import API

- Code generation
-Metamodels for Petri net types

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

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

CREA
TE                   SA
LOAD
FETCH
Limitations

- Static code generation API, w.r.t. the metamodels
- Every metamodel update implies a code regeneration
- 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

PNML Framework

How is it built?

Application examples

Conclusion / Resources

Limitations / Improvement

Resources
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://www.pnmlweb.org (Fall 2009)