Software and Systems Engineering – High-level Petri Nets

Part 2: Transfer Format

International Standard ISO/IEC 15909-2
WD Version 0.9.0

June 23, 2005

Contents

Introduction

This International Standard is Part 2 of a multi-part standard concerned with defining a modelling language and its transfer format, known as *High-level Petri Nets*. Part 1 of this International Standard provides the mathematical definition of High-level Petri Nets, called the semantic model, the graphical form of the technique, known as High-level Petri Net Graphs (HLPNGs), and its mapping to the semantic model. Part 1 also introduces some common notational conventions for HLPNGs 0.0.0.1

Part 2 of this International Standard defines a transfer format for High-level Petri Nets in order to support the exchange of High-level Petri Nets among different tools. This format is called the *Petri Net Markup Language* (*PNML*). Since there are many different versions of Petri nets in addition to High-level Petri Nets, this Standard defines the *Core Concepts* of all Petri net types along with an XML syntax, which can be used for exchanging any kind of Petri Net. Based on this PNML Core Model, Part 2 also defines the transfer syntax for the two versions of Petri Nets that are defined in Part 1 of this International Standard. Place/Transition Nets and High-level Petri Nets. Accordingly, conformance to this International Standard is in three levels: The basic level of conformance is to the PNML Core Model. The other levels are according to the particular type, which is conformance to Place/Transition Nets and conformance to High-Level Petri Nets. 0.0.0.2

An addendum to Part 1 of this International Standard defines Well-formed Nets as a subclass of High-level Petri Nets, which uses a restricted set of operators. 0.0.0.3

Editor's note 1: The name 'WFN' for this class is still under discussion and must be inserted here, once this name is accepted. A proposal is 'Structured Class Nets', but it is not yet replaced in this text.

Editor's note 2: An overview of the different Clauses of this International Standard will be inserted here later.

1 Scope

Editor's note 1.1: There was a proposal to structure, the Scope as follows:

- Purpose
- Field of Application
- Audience

There was no time to include that yet. But, this restructuring should be easily possible in the next version.

This International Standard defines a transfer format for Petri nets. It is based on XML and is called the *Petri Net Markup Language* (*PNML*). PNML facilitates the exchange of Petri nets among different Petri net tools. 1.0.0.1

There are many different variants of Petri nets. Place/Transition Nets and High-level Petri Nets as defined in Part 1 of this International Standard are just two versions. Some possible extensions and special Petri net types will be defined in Part 3 of this International Standard. 1.0.0.2

One objective of PNML is to support the exchange of Petri Nets among different tools, even if the tools support slightly different versions of Petri Nets. Features of Petri nets not known to other tools can be easily ignored without losing all information associated with the original net. 1.0.0.3

Therefore, Part 2 of this International Standard defines the core concepts that are common to all kinds of Petri nets. These concepts are captured in the *PNML Core Model*, which is defined as a meta model in UML notation. For each concept in this meta model, this International Standard defines a precise XML representation. 1.0.0.4

Based on the PNML Core Model, this International Standard defines a transfer format for the two versions of Petri nets defined in Part 1 of this International Standard: Place/Transition Nets and for High-level Petri Nets. Well-Formed Petri Nets are defined as a special case of High-level Petri Nets. 1.0.0.5

The definition of other features of Petri nets and of other versions of Petri nets, such as those incorporating time, will be defined in Part 3 of this International Standard. 1.0.0.6

Editor's note 1.2: Some details on Well-formed Nets (as well as their upcoming name) must be inserted here.

Editor's note 1.3: In Helsinki, it was decided not to define the mechanism for defining Petri Net Types (Petri Net Type Definitions PNTDs) in Part 2 of this standard. The types will be defined by merging UML packages here. The mechanism for defining Petri Net Types could be defined in Part 3.

The PNML Core Model as defined in this International Standard also includes a simple concept for structuring Petri nets into different pages. More complex structuring mechanisms and a module concept are not part of this International Standard, but will be addressed in Part 3 of this International Standard. 1.0.0.7

Editor's note 1.4: In Helsinki, it was decided to include the page concept, but not to include the module concept to this part of the standard.

2 Conformance

There are different levels of Conformance. All conformance levels impose additional conditions on XML documents. 2.0.0.1

An XML document is conformant to the PNML Core Model if it meets the definitions of Clauses 5.2 (PNML Core Model) and 6.1 (its XML representation) – such a document is called a PNML Document or a Petri Net Document. A Petri net tool is conformant to the PNML Core Model, if it can import all PNML Documents, and if it can export all Petri nets to a PNML Document. 2.0.0.2

A PNML Document is a conformant Place/Transition Net, if it meets the additional restrictions of Clause 5.3.1 – such a document is called a PNML Place/Transition Net Document. A Petri net tool is conformant to PNML Place/Transition Nets, if it can import all PNML Place/Transition Net Documents, and if it can export all Place/Transition Net to a PNML Place/Transition Net Document. 2.0.0.3

A PNML Document is a conformant High-level Petri Nets, if it meets the additional restrictions of Clause 5.3.2 – such a document is called a PNML High-level Petri Net Document. A Petri net tool is conformant to PNML High-level Petri Nets, if it can import all PNML High-level Petri Net Documents, and if it can export all High-level Petri Nets to a PNML High-level Petri Net Document. 2.0.0.4

3 Normative References

The following references are indispensable for the application of this International Standard. 3.0.0.1

- CSS: Cascading Style Sheets, level 2 revision 1, CSS 2.1 Specification. W3C Candidate Recommendation 25 February 2004.
- High-level Petri Nets: ISO/IEC 15909:2004 (E) Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation. First edition, December 1, 2004, ISO/IEC.
- JPEG: ISO/IEC 15444: Information technology – JPEG 2000 image coding system.
- MathML: Mathematical Markup Language (MathML) Version 2.0 (Second Edition). W3C Recommendation 21 October 2003.
- PNG: Portable Network Graphics (PNG) Specification (Second Edition). W3C Recommendation 10 November 2003.
  Also: ISO/IEC 15948:2003 (E) Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification.
- UML 2.0: Unified Modeling Language: Superstructure, version 2.0, Revised Final Adopted Specification. OMG ptc/04-10-02, October 8, 2004 OMG.

Editor's note 3.1: In Helsinki, we decided to use some notation from UML 2.0, which should be compatible with WG 19's move towards UML 2.0

- XML 1.0: Extensible Markup Language (XML), Version 1.0 (Third Edition); W3C Recommendation 04 February 2004.
- XMLSchema Datatypes: XMLSchema Part 2: Datatypes (Second Edition); W3C Recommendation 28 October 2004

Editor's note 3.2: It is not clear yet, whether we will be needing references to OCL, XSLT, SVG or RELAX NG

4 Terms and Definitions

For the purpose of this International Standard, the following terms and definitions apply. 4.0.0.1

4.1 Definitions from Part 1, ISO/IEC 15909-1

The following terms and definitions are adopted from Part 1 of this International Standard. 4.1.0.1

Arc, Arc Annotation, Arity, Basis Set, Declaration, High-level Petri Net, High-level Petri Net Graph, Many Sorted Algebra, Marking (of a net), Marking of a place, Multiset, Net, Net Graph, Node, Operator, Petri Net, Place/Transition Net, Place, Place Type, Signature, Sort, Argument Sort, Range Sort, Term, Closed Term (Ground Term), Transition, Transition Condition, Type. 4.1.0.2

4.2 Definitions for Part 2

4.2.1 Annotation A Label represented as text near to the Object it is associated with.

Editor's note 4.1: I am not sure when terms should be capitalized. If all occurrences of these terms need to be capitalized, the text looks a bit awkward. Are there any rules? If not, can we just emphasis these terms? In the next version, this will be made consistent.

4.2.2 Attribute A Label that governs the form or shape of the Object it is associated with. In contrast to an Annotation, the Attribute is not shown as a text.

4.2.3 Graphical Information The information required to define the graphical appearance of Objects and Labels of a Net Graph. This can be the position, size, line colour, fill colour, font or the line width.

4.2.4 Global Label A Label associated with the Net Graph itself, rather than with an Object of a Net Graph.

4.2.5 Label Information associated with the Net Graph or one of its Objects.

4.2.6 Meta Model A model defining the concepts and their relations for some modelling notation. In this International Standard, we use UML diagrams – in particular class diagrams – for defining the core concepts of PNML (PNML Core Model) as well as the concepts specific to some Petri net types.

4.2.7 Object (of a Net Graph) The Arcs, Nodes, Reference Nodes, and the Pages of a Net Graph.

4.2.8 Page A structuring mechanism used to split large Net Graphs into smaller parts, which are also the units of the net to be printed.

4.2.9 PNML Core Model The Meta Model defining the basic concepts and structure of Petri Net models that are common to all versions of Petri nets.

4.2.10 PNML Document (Petri Net Document) A document that contains one or more Net Graphs.

4.2.11 PNML High-level Document A PNML Document that contains one or more Net Graphs, where all Petri nets conform to High-level Petri Nets.

4.2.12 PNML Place/Transition Net Document A PNML Document that contains one or more Petri Net Graphs, where all Petri nets conform to Place/Transition Nets.

4.2.13 Reference Node A Node of a Net Graph that is a representative of another Node possibly defined on another Page of the Net Graph. Cyclic references, however, are forbidden. A Reference Node does not carry any semantical information itself. Rather, it is a pointer to the Node that is being referenced.

4.2.14 Reference Place A reference node that is a place; it must refer to either another reference place or to a place.

4.2.15 Reference Transition A reference node that is a transition; it must refer to either another reference transition or to a transition.

4.2.16 Source Node The node associated with the start of an arc.

4.2.17 Target Node The node associated with the end of an arc.

4.2.18 Tool Specific Information Information associated with Objects of a Net Graph or with the Net Graph itself that is specific to a particular tool and is not meant to be used by other tools.

4.3 Abbreviations

- HLPNG: High-level Petri Net Graph
- WFN: Well-Formed Petri Net
- UML: Unified Modelling Language
- XML: eXtensible Markup Language
- PNML: Petri Net Markup Language
- P/T Net: Place/Transition Net
- CSS: Cascading Stylesheets

Editor's note 4.2: This list needs to be fixed in the end.

5 Concepts

5.1 General Principles

This International Standard defines a transfer format for High-level Petri Net Graphs and Place/-Transition Nets as defined in Part 1 of this Standard. This transfer format has been designed to be extensible and open for future variants of Petri nets and possibly for other use, such as the transfer of results associated with the analysis of Petri nets, e.g., reachability graphs. In order to obtain this flexibility, the transfer format considers a Petri net as a labelled directed graph, where all type specific information of the net is represented in labels. A label may be associated with a node or an arc of the net or with the net itself. This basic structure is defined in the *PNML Core Model*, which is defined in Clause 5.2. 5.1.0.1

The PNML Core Model is presented using UML class diagrams. Note that these UML diagrams do not define the concrete XML syntax for PNML Documents. Clause 6 defines the mapping of the PNML Core Model elements to XML syntax. 5.1.0.2

Editor's note 5.1: We do not use RELAX NG for this mapping anymore (though we give a definition of the syntax of PNML Documents in RELAX NG). We give an explicit translation for each element. Suggestions for an appropriate technology are welcome.

The PNML Core Model imposes no restrictions on labels. Therefore, the PNML Core Model can represent any kind of Petri net. Due to this generality of the PNML Core Model, there can be even models that do not correspond to a Petri net at all. For a concrete version of Petri nets, the legal labels will be defined by extending the PNML Core Model with another Meta Model that exactly defines the legal labels of this type. 5.1.0.3

Technically, the PNML Core Model is a UML package, and there are additional UML packages for the different Petri net types that depend on the PNML Core Model package. Part 2 of this International Standard defines a package for Place/Transition Nets and for High-level Petri Net Graphs. High-level Net Graphs subsume Place/Transition Nets, which means that any legal label of a Place/Transition Net may occur also in a High-level Petri Net Graph. 5.1.0.4

Editor's note 5.2: We do not explicitly mention the Type and Feature Definitions anymore. These concepts will be introduced in Part 3. Also we do not have the Conventions anymore.

Figure 1 gives an overview of the different meta models defined and on their dependencies. The package PNML Core Model defines the basic structure of Petri nets; this structure will be extended by the package for the two types. The PNML Core Model will be defined in Clause 5.2, the package PT-Net will be defined in Clause 5.3.1, and the package HLPNG will be defined in Clause 5.3.2. In Clause 6 we will show how the concepts defined in these packages are mapped to concrete XML syntax. 5.1.0.5

Part 2 of this International Standard introduces some concepts that are not defined in Part 1. In Part 1, there are no concepts of pages and reference nodes, and there are no concepts for defining the graphical appearance of Petri nets. Since this is important information when it comes to the exchange of Petri nets between different tools, its was necessary to introduced these concepts here. Since, this information does not carry any semantics, these concepts are compatible with Part 1 of this International Standard. 5.1.0.6

5.2 PNML Core Model

Figure 2 shows the PNML Core Model as a UML class diagram. This diagram will be discussed in the following Clauses. 5.2.0.1

5.2.1 Petri net documents, Petri nets, and objects.

A document that meets the requirements of PNML is called a *Petri net document* (PetriNetDoc). It may contain several *Petri Nets*. Each Petri Net includes a unique identifier and type. The type is a URL referring to the name of the package with its definition. 5.2.1.1

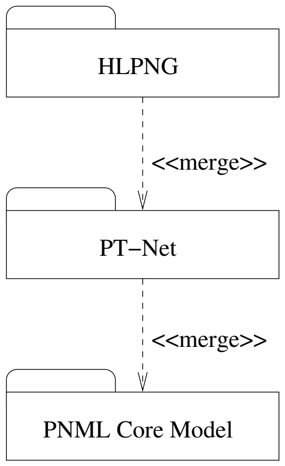


Figure 1: Overview on the UML packages of PNML

Editor's note 5.3: We require that every object and every annotation has a graphical information (at least the position). This it not strictly necessary; but, I suggest to require it anyway. Because not having positions, will make it quite hard to exchange nets among graphical tools.

Editor's note 5.5: We need a naming convention for the packages. Would it be something `ISO-IEC.PNML.PT-System`? Up to now, we used a URL referring to the RELAX NG grammar. But with using UML technology, this is deprecated.

We could use also the Eclipse naming conventions for PlugIns and packages as proposed by AFNOR. E. g. `org.iso_iec.15909_2.pnml.ptnets_1.0.0`. The corresponding jar files of the API could be named accordingly.

A Petri net consists of a single top-level *page* that in turn consists of several *objects*. These objects, basically, represent the graph structure of the Petri net. Each object within a Petri net document has a unique *identifier*, which can be used to refer to this object. Moreover, each object may be equipped with graphical information defining its position, size, colour, shape and other attributes on its graphical appearance. The precise graphical information that can or must be provided for an object depends on the particular type of object (see Clause 5.2.4). 5.2.1.2

An object is a *place*, a *transition* or an *arc*. For convenience, a place or a transition is generalized to a *node*, which can be connected by arcs. 5.2.1.3

Note that it is legal to have an arc from a place to a place or from a transition to a transition according to the PNML Core Model. The reason is that there are versions of Petri nets that support such arcs. If a Petri net type does not support such arcs, this restriction will be defined in the particular package defining this type. 5.2.1.4

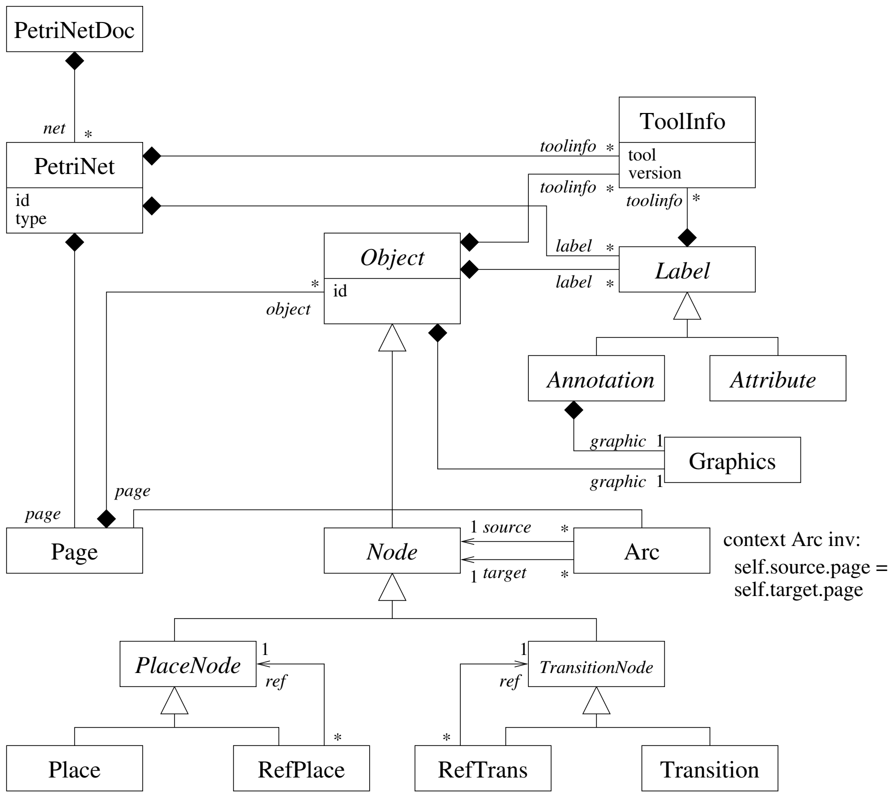


Figure 2: The PNML Core Model

Editor's note 5.4: The OCL expression in this diagram expresses that the nodes connected by an arc must be on the same page. It should be discussed whether we would should use these OCL expressions in this Standard or not. In either case, the corresponding condition will be stated also in the text.

5.2.2 Pages and reference nodes.

Three other kinds of objects are used for structuring a Petri net: *pages*, *reference places*, and *reference transitions*. As mentioned above, a page may contain objects; since a page is an object itself, a page may even contain other pages, which defines a hierarchy of subpages. 5.2.2.1

This International Standard requires that an arc must connect nodes on the same page only. The reason for this requirement is that arcs connecting nodes on different pages cannot be drawn graphically on a single page. 5.2.2.2

Editor's note 5.6: Just to show how this condition could be expressed in OCL, I included the corresponding OCL expression to the UML diagram.

In order to connect two nodes on different pages by an arc, a representative of one of the two 5.2.2.3

nodes is drawn on the same page as the other node. Then, this representative may be connected with the other nodes by an arc. This representative is called a *reference node*, because it has a reference to the node it represents (see Clause 5.2.2). Note that a reference place must refer to a place or a reference place, and a reference transition must refer to a reference transition or a transition. Cyclic references among reference nodes are not allowed.

Editor's note 5.7: The condition of cycle free references can – to the best of my knowledge – not be expressed in the current version of OCL.

5.2.3 Labels.

In order to assign further meaning to an object, each object may have *labels*. Typically, a label represents the name of a node, the initial marking of a place, the transition condition, or an arc annotation. In addition, the Petri net itself, resp. its pages may have some labels. These are called *global labels*. For example, the package HLPNG defines declarations of types as global labels of a high-level Petri net. 5.2.3.1

This International Standard distinguishes two kinds of labels: *annotations* and *attributes*. An annotation comprises information that is typically displayed as text near the corresponding object. Examples of annotations are names, initial markings, arc annotations, transition conditions, and timing or stochastic information. 5.2.3.2

Editor's note 5.8: There was a comment that text should be defined here. In XML documents text will be represented as XML PCDATA elements, i.e. a sequence of characters. This is defined in Clause 6, where we deal with the mapping of the concepts of the meta model to XML syntax.

In contrast, an attribute is not displayed as text near the corresponding object. Rather, an attribute has an effect on the shape or colour of the corresponding object. For example, an attribute *arc type* could have domain `normal`, `read`, `inhibitor`, `reset`. This International Standard, however, does not mandate the effect on the graphical appearance of an attribute. 5.2.3.3

Note that Label, Annotation and Attribute are abstract classes in the PNML Core Model, which means that the PNML Core Model does not define concrete labels, annotations, and attributes. The concrete labels will be defined in the meta models for the concrete types of Petri nets (see Clause 5.3). 5.2.3.4

In order to support the exchange of information among tools that have different textual representation for the same concepts (i. e. if they have different concrete syntax), there are two ways for representing the information within an annotation: textually in some concrete syntax and structurally as an abstract syntax tree (see Clause 6.1.2 for details). 5.2.3.5

Editor's note 5.9: There could be a UML diagram that defines this general structure of labels. Since it was decided not to include the definition of the type definition interface to Part 2, this is not necessary for Part 2.

Editor's note 5.10: Right now, it is legal to have the abstract syntax tree only. We could require that there must always be a textual representation. This needs to be discussed.

Note that reference nodes may have labels but these labels can only specify information about their appearance. They cannot specify any information about the referenced node itself, which already has its own labels for this purpose. 5.2.3.6

5.2.4 Graphical information

Graphical information can be associated with each object and each annotation. For a node, this information includes its position; for an arc, it includes a list of positions that define intermediate points of the arc; for an object's annotation, it includes its relative position with respect to the corresponding object; and for an annotation of the net itself, the position is absolute. There can be further information concerning the size, colour and shape of nodes or arcs, or concerning the colour, font and font size of labels (see Clauses 6.1.3 for more information). 5.2.4.1

Editor's note 5.11: There was a proposal of a much more detailed meta model for the graphical information on the PNX Mailing list from Celso Gonzalez from Canada, which covers most of the details.

For lack of time, this is not yet included here.

5.2.5 Tool specific information (ToolInfo)

For some tools, it might be necessary to store tool specific information, which is not meant to be used by other tools. In order to store this information, *tool specific information* may be associated with each object and each label. Its internal structure depends on the tool and is not specified by PNML. PNML provides a mechanism for clearly marking tool specific information along with the name and the version of the tool adding this information. Therefore, other tools can easily ignore it, and adding tool specific information will never compromise a Petri net document. 5.2.5.1

The same object may be tagged with tool specific information from different tools. This way, the same document can be used and changed by different tools at the same time. The intention is that another tool should not change or delete the information added by another tool as long as the corresponding object is not deleted. Moreover, the tool specific information should be self contained and not refer to other objects of the net because the deletion of other objects 5.2.5.2

by a different tool might make this reference invalid and leave the tool specific information inconsistent. This use of the tool specific information is strongly recommended; however, it is not normative!

5.3 Petri Net Type Meta Models

This Clause defines meta models for two versions of Petri nets: Place/Transition Net and High-Level Petri Net Graphs (HLPNGs) as defined in Part 1 of this International Standard. These meta models define the labels of the respective Petri net type. 5.3.0.3

Though HLPNGs conceptually capture and generalize Place/Transition Nets, there are differences in syntax. Of course, there are mappings from the syntax of the concepts of Place/Transition Nets to HLPNGs. But, in order not to force tools that support Place/Transition Nets only to use the syntax of HLPNGs, we introduce a simple syntax for Place/Transition Nets. The transfer syntax for HLPNGs, however, also supports the syntax of Place/Transition Nets (cf. Fig. 1). 5.3.0.4

Editor's note 5.12: This was decided in Helsinki. It was also decided to include some guidelines on how HLPNG tools should deal with the Place/Transition Net concepts. These guidelines are not yet included to this text.

AFNOR suggested that they could make a proposal.

Note that Well-formed Petri Nets, which are defined in an Addendum to Part 1 of this International Standard, are a special version of HLPNGs with a restricted set of built-in types and operations (see Addendum to Part 1). Since this is a semantical issue, it is hard to capture this difference syntactically, which is the reason for not making it a separate (syntactical) Petri net type. 5.3.0.5

Editor's note 5.13: The problem is that we do not know yet how to define this restriction of MathML expressions. So this is a semantical restriction as defined in the Addendum to Part 1.

5.3.1 Place/Transition Nets

This Clause defines the meta model for Place/Transition Nets in terms of a UML package *PT-Net*. 5.3.1.1

A Place/Transition Net is a Net Graph, where the places and transitions have names, and where each place is labelled with a natural number representing the Initial Marking, and each arc is labelled with a non-zero natural number representing the Arc Annotation. The meaning of these labels are defined in Part 1 of this International Standard. 5.3.1.2

Figure 3 shows the package PT-Net. The OCL expression on the lower left side expresses that in Place/Transition Nets arcs must not connect places to places or transitions to transitions. 5.3.1.3

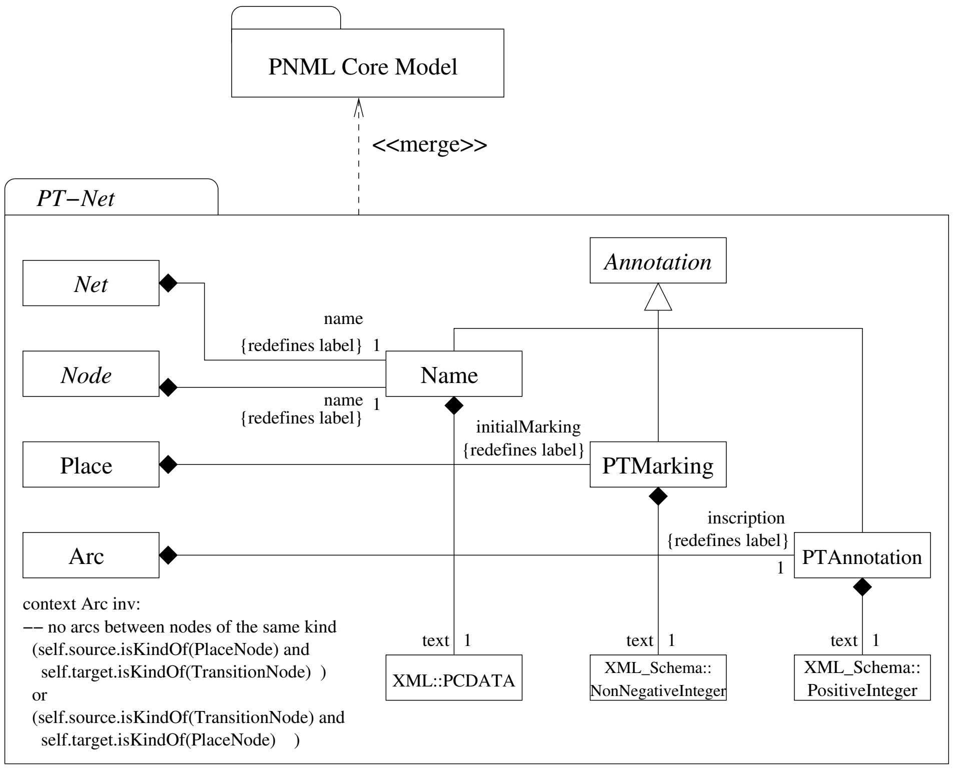


Figure 3: The package PT-Net

Editor's note 5.14: Maybe the OCL condition will be deleted in the end. This example should give an impression of an extra condition expressed in OCL. Even if the OCL condition is omitted from the UML diagram, the conditions in the text above will stay and are mandatory.

This package defines that each node of a P/T-Net must have an Annotation *Name* that consists of *text* (in addition to the standard information for annotations such as graphical and tool specific information, which are defined in the PNML Core Model). The text will be represented as XML *PCDATA*, which is expressed by referring to the class XML::PCDATA (see Clause 6.2). 5.3.1.4

In addition, the package defines that each Place must have an Annotation *PTMarking* that represents a natural number. Technically, the type and the representation of the contents of this label is defined by referring to the data type *NonNegativeInteger* of XMLSchema. 5.3.1.5

Each Arc must have an Annotation *PTAnnotation* that consists of a non-zero natural number, where the representation of this label is defined by the data type *PositiveInteger* of XMLSchema. 5.3.1.6

The UML Meta Models do not yet fix all details of the XML syntax for these elements. The exact mapping to XML syntax will be defined in Clause 6.2. 5.3.1.7

Note that the only classes defined here are Name, PTMarking, and PTAnnotation. The classes Node, Place, Arc, and Annotation come from the package PNML Core Model. They are imported here in order to define the concrete Labels attached to these Nodes in this particular type, i.e. in Place/Transition Nets. Likewise, the classed prefixed with XML and XMLSchema are 5.3.1.8

defined in other packages in order to refer to standard concepts of XML and standard data types of XMLSchema.

Editor's note 5.15: Eventually, the class Name, PTMarking, and PTAnnotation could be imported from a package of standard Labels. But, this is left to Part 3

5.3.2 High-Level Petri Net Graphs

This Clause defines the meta model for High-Level Petri Net Graphs. 5.3.2.1

A HLPNG is a Petri Net Graph that is equipped with Declarations that defines Types, Functions, and Variables that are the basis for defining Terms. The Declarations can be Annotations of the Petri Net Graph itself or of some of its Pages; the Types can be Annotations of the Place; a place can also have an Annotation with its Initial Marking which is an Expression for some multiset; an Arc Annotations is some MathML expression; a Transition can have a Condition as its Annotation, which is a MathML expression of type boolean. These concepts and their semantics are precisely defined in Part 1 of this International Standard. 5.3.2.2

The package defining the concepts of HLPNGs is shown in Fig. 4. Where the contents of these Annotations is XML in MathML syntax, which is indicated by classes prefixed with MathML. 5.3.2.3

Editor's note 5.16: In Helsinki, it was decided to use MathML. Unfortunately, we do not know yet, how to make sure that these MathML expressions are legal at their position in a syntactical way.

Note that this model defines an abstract syntax for all HLPNG concepts only. In order to allow tools to store some concrete text, the Annotation may also consist of text, which should be the same expression in some concrete syntax of some tool. The concrete syntax, however, will not be defined in this International Standard. 5.3.2.4

In addition to the Annotations defined above, the package for HLPNGs requires that arcs must not connect two places and must not connect two transitions; this however, is a property inherited from Place/Transition Nets. Therefore, it is not necessary to include this condition here. 5.3.2.5

5.3.3 BuiltIn Sorts and Functions

Editor's note 5.17: I left the text of this section from an earlier version in this document, so it is not consistent anymore. Due to the use of MathML we cannot define our own built-in sorts anymore. The problem with MathML is that there are some types that do not make sense in Petri nets (complex) and some types (such as Strings) do not exist in MathML. Moreover, we cannot characterize an appropriate subset of MathML syntactically. This is an open issue.

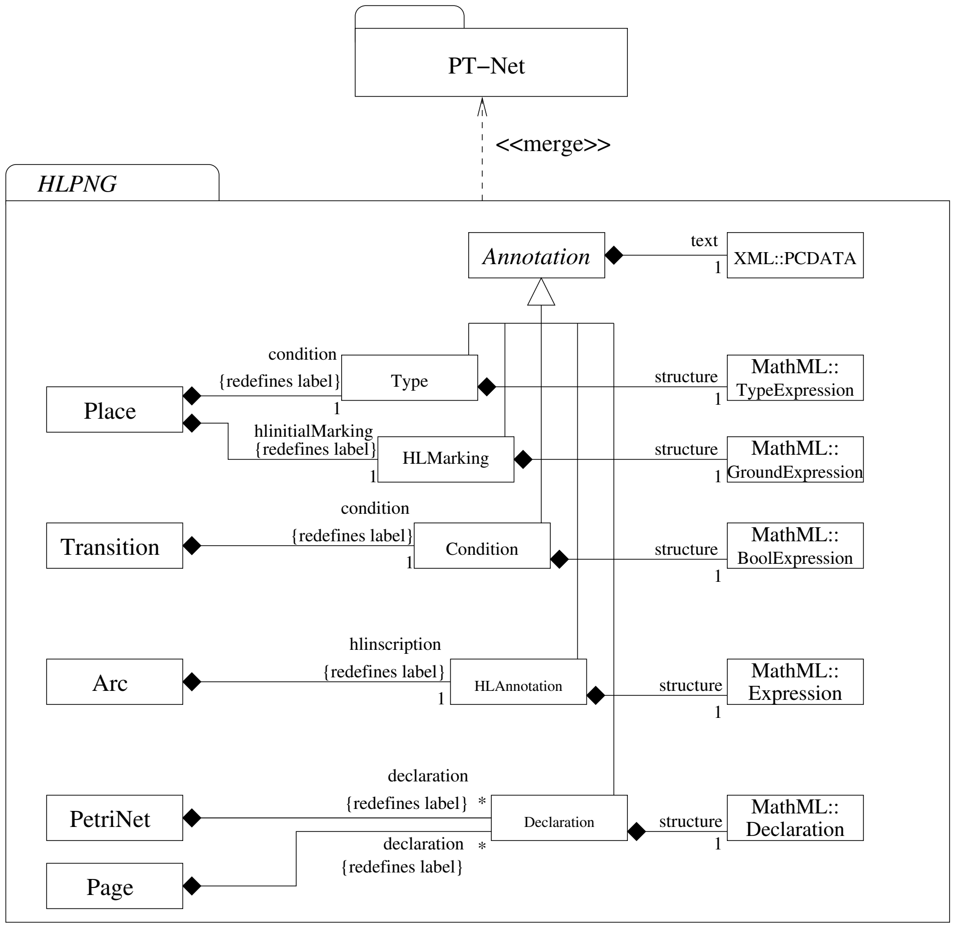


Figure 4: The package HLPNG

HLPNGs and WFN are defined using the very same Meta Model as defined in Clause 5.3.2. The only difference is in the builtin Sorts, the builtin Functions, and the Sorts that can be defined by the User. 5.3.3.1

Editor's note 5.18: The Type Model for HLPNGs is an abstract version of the RELAX NG definitions circulated in the previous draft. Moreover, it seems to match the AFNOR proposal (cf. [15]).

For defining HLPNGs and WFNs we must define the builtin Sorts and Functions only. These will be defined blow: 5.3.3.2

Editor's note 5.19: This is a first proposal that is based on the Bologna discussions.

HLPNGs 5.3.3.3

- boolean with all boolean operations: and, or, not, implication, equality.
- integer with addition, subtraction, multiplication, div, and mod and the operations for comparison (=, <=, <, >, >=, <>).

Editor's note 5.20: At the Bologna the question was raised how to deal with the finiteness of integers. Should this standard give a particular definition or do we leave it open.

- ranges of integers with comparison (=, <=, <, >, >=, <>).
- explicit enumerations with successor and predecessor operation and with equality and comparison operations.
- strings with concatenation and comparison operations.
- For each Sort, there are implicit operations, converting a sequence of elements of this Sort to a multiset [a1, a2, a3].

Editor's note 5.21: Products are included explicitly in the Meta Model; so it is not necessary to include this here

Editor's note 5.22: At the Bologna meeting List and Disjoint Sets were discussed. As far as I remember, it was not yet decided whether to include them or not.

Editor's note 5.23:

- The following text was copied verbatim from a AFNOR proposal. It is not yet polished and adapted to this context.
- It is important to notice that WFN name has changed to Structured Class Net.
- Shall we really put in the following definitions since they have been already fully detailed in Annex B2 of part 1? May be should we just put a reference to Annex B2.

Structured Class Nets 5.3.3.4

Structured Class Nets is a Class of High-level Petri Net Graphs. It is defined in Annex B.2 of the Part 1 of this International Standard. This class places restrictions on the many-sorted algebra of HLPNGs. The carriers of the algebra are finite, and only a small set of functions, as defined below, are allowed. The carriers in the algebra are referred to as types or domains. Basic types are defined and then further (structured) types are built from the basic types. Basic types can be ordered-circular or unordered.

The following basic types and their usual operators are required:

- The Booleans, $Boolean$, with unary function ‘not’, and binary functions: ‘equals’ ($=$), ‘not equals’ ($\neq$), and usual connectives: ‘and’ and ‘or’;
- Any restricted finite range of Integers (e.g. $\{0, 1, 2, 3, 4\}$, also written [0,4]) with the Identity function;
- Enumerated types (i.e. types that can be placed in one to one correspondence with a finite integer range);
- binary functions ’equal’ ($=$) and ’not equal’ ($\neq$) are defined on both ordered-circular and unordered types.
- Successor and predecessor unary functions are defined on ordered-circular types;
- binary operator ’is element of’ (noted $\in$), $x \in C_i$ means that x is an element of subclass C_i ;

The following structured types built from the above basic types are also required:

- Cartesian products of basic types ; recursive definitions of product types is allowed;
- Multisets over basic types, product and set types.

The following functions on basic types can be combined with the product operator: $=$, $\neq$, identity. No user-defined functions are allowed in Structured Class Nets.

The type of a Place in a Structured Class Nets is restricted to basic types, products and set types.

6 PNML Syntax

Clause 5 defines the concepts of the PNML Core Model and the concepts of Place/Transition Nets and High-level Petri Nets. Clause 5, however, does not define a precise XML syntax for representing these concepts. The precise syntax of the PNML Core Model and the two Petri net types will be defined in this Clause. 6.0.3.1

Clause 6.1 defines the format of general PNML Documents. Clause 6.2 defines the format for Place/Transition Nets and High-level Petri Nets. In fact, it gives general rules, how the concepts of a package defining some Petri net type are mapped to XML. 6.0.3.2

Editor’s note 6.1: Here, we give an explicit mapping from the PNML concepts to XML syntax. This could be done by some UML/XML-technology. This is an open issue. I am not sure whether we should deal with that in Part 2. Maybe, this could be covered in Part 3 – in the context of the PNML API.

Editor's note 6.2: Some of the element names or attribute names chosen below, might be a bit misleading. But, for sake of compatibility with existing implementations, we should have very good reasons for changing them. Incompatibilities will inevitably result, when we change PNML keywords.

6.1 PNML Documents

The mapping of the PNML Core Model concepts to XML syntax is defined for each class of the PNML Core Model diagram. 6.1.0.3

6.1.1 PNML elements.

Each concrete class[1] of the PNML Core Model is mapped to an XML element. The translation of these classes along with the attributes and their data types is given in Table 1. These XML elements are the *keywords* of PNML and are called *PNML elements* for short. For each PNML element, the compositions of the PNML Core Model define in which elements it may occur as a child element. 6.1.1.1

The data type ID in Table 1 refers to a set of unique identifiers within the PNML Document. The data type IDRef refers to the set of all identifiers occurring in the document, i. e. they are meant as references to identifiers. A reference at some particular position, however, is restricted to objects of a particular type – as defined in the PNML Core Model. For instance, the attribute `ref` of a reference place must refer to a place or a reference place of the same net. The set to which a reference is restricted, is indicated in the table (e. g. for a reference place, the attribute `ref` should refer to the `id` of a Place or a RefPlace). 6.1.1.2

Editor's note 6.3: Technically, indicating this restriction is not necessary anymore because it is now captured in the UML diagrams. But, for readability reasons, I left this information in this table.

6.1.2 Labels

There are no PNML elements for labels because the PNML Core Model does not define any concrete labels. For concrete Petri net types, such as Place/Transition Nets and High-level Petri nets, the corresponding packages define these labels. 6.1.2.1

In general PNML Documents, any XML element that is not defined in the PNML Core Model (i. e. not occurring in Table 1) 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>` could represent the name of an object, and `<inscription>` an arc annotation. 6.1.2.2

A legal element for a label must contain at least one of the two following elements, which 6.1.2.3

[1] A class in a UML diagram is concrete if its name is not displayed in italics.

Table 1: Translation of the PNML Core Model into PNML elements

Class	XML element	XML Attributes
PetriNetDoc	`<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>`	

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 [2].

Editor's note 6.4: Right now, it is legal to have the abstract syntax tree only. We could require that there must always be a textual representation. This needs to be discussed.

An optional PNML `<graphics>` element defines its graphical appearance; and optional PNML `<toolspecific>` elements may add tool specific information to the label. 6.1.2.4

6.1.3 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 2 shows the elements which may occur within the `<graphics>` element. 6.1.3.1

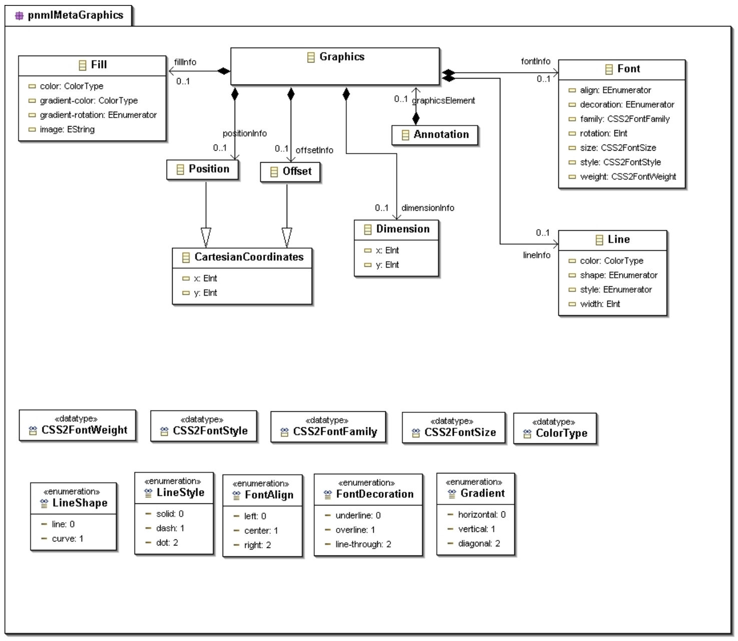
Editor's note 6.5: Actually, the different graphical elements could be captured in the UML diagram. Then, the mapping of graphics will be more straight-forward. This is an open issue (not difficult, but not done for lack of time).

[2] In order to be compatible with earlier versions of PNML, the text element `<value>` may occur alternatively to the `<text>` `<structure>` pair.

Table 2: Possible child elements of the <graphics> element

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

There is a first proposal from AFNOR:



But, there should be even different graphics classes in order to formalize in UML which kind of graphical information could go to which kind of Object. There should be at least three classes: NodeGraphics, ArcGraphics, and AnnotationGraphics.

Editor's note 6.6: There was a proposal to attach graphical information to a net at the PNML workshop at HUT. Since we decided that every net should have at least on page containing all other objects, this is not necessary anymore. Therefore, there is no graphical information for the complete net. This goes to the top-level page.

The <position> element defines an absolute position and is required for each node, whereas 6.1.3.2

the <offset> element defines a relative position and is required for each annotation. Each absolute or relative position refers to Cartesian coordinates (x, y). As for many graphical tools, the x-axis runs from left to right and the y-axis from top to bottom.

Editor's note 6.7: There was a discussion that it should be possible to give an absolute position for labels. The result, however, was that we should not have them.

The other sub-elements of <graphics> are optional. For an arc, the (possibly empty) sequence of <position> elements defines its intermediate points (bend points). 6.1.3.3

The only required graphical information for a Node and Page is the position; the graphical information required for Annotations is the offset. Note that, for arcs, it is not necessary to give a position, because these are intermediate positions only. The start and end is defined by the source and target node. 6.1.3.4

Table 3 defines the attributes for each graphical element defined in Table 2. 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. 6.1.3.5

Table 3: 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 <position> element defines the absolute position of a node on a page. The <offset> element defines the position of an annotation relative to the position of the object – if it is a global annotation, it defines the absolute position on that page. 6.1.3.6

For an arc, there may be a (possibly empty) list of <position> elements. These elements define intermediate points of the arc. Altogether, the arc is a path from the source node of the arc to the destination node of the arc via the intermediate points. Depending on the value of attribute `shape` of element <line>, the path is displayed as a polygon or as a (quadratic) Bezier curve, where the points act as line connectors or Bezier control points. 6.1.3.7

The <dimension> element defines the height and the width of a Node. 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. 6.1.3.8

Editor's note 6.8: Note that there is no need for explicitly stating that a place is displayed as an ellipse and a transition as a rectangle in the XML syntax. This information comes from the fact that the node is a place or a transition. The only way to override this is by using attribute images in the graphical element fill.

If the dimension of an element is missing, each tool is free to use its own default value for the dimensions. 6.1.3.9

The two elements <fill> and <line> define the interior and outline colours of the corresponding element. The value assigned to a colour attribute must be a RGB value or a predefined colour as defined by CSS2. When the attribute `gradient-color` is defined, the fill colour continuously varies from color to gradient-color. The additional attribute `gradient-rotation` defines the orientation of the gradient. If the attribute `image` is defined, the node is displayed as the image at the specified URI, which must be a graphics file in JPEG or PNG format. In this case, all other attributes of <fill> and <line> are ignored. 6.1.3.10

For an annotation, the <font> element defines the font used to display the text of the label. The complete description of possible values of the different attributes can be found in the CSS2 specification. Additionally, the `align` attribute defines the justification of the text with respect to the label coordinates, and the `rotation` attribute defines a clockwise rotation of the text. 6.1.3.11

6.1.4 Example

Editor's note 6.9: At the Brisbane meeting, there was a proposal that there should be a better example. I do agree to that. unfortunately, I had no time for producing a better example. But, I hope that with the support from others, we can have a better example in the next version of the Working Draft. This could be a running example.

In order to illustrate the structure of a PNML Document, we give an example PNML document representing the Petri net shown in Fig. 5, which actually is a Place/Transition Net. Listing 1 shows the corresponding PNML Document. It is a straight-forward translation, where there are labels for the names of objects, for the initial markings, and for arc annotation. 6.1.4.1

Note that a tool would typically display the initial marking as a textual label. But, we assume 6.1.4.2

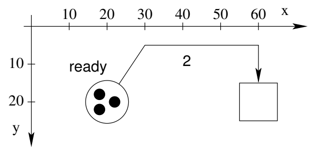


Figure 5: A simple Place/Transition Net

that this net is viewed by an imaginary tool *PN4all*; in the PNML Document, this tool has added some tool specific information in the annotation for the initial marking. We assume that the imaginary tool interprets its own tool specific information in such a way that it shows the tokens at individual positions as given in the elements `<tokenposition>`

Since there is no information on the dimensions in this example (in order to fit the listing to a single page), the tool has chosen its default dimensions for the place and the transition. 6.1.4.3

Editor's note 6.10: In Part 1 there were no pages and reference nodes. In order to use the semantics of Part 1, we give a 'semantics' of net with pages of reference in terms of a net without. This could be done by simply flattening the net: Omit pages and merge all reference nodes to the node they refer to (by omitting the labels of the reference nodes). This way, one gets a net with Places, Transitions and Arcs only. The details can be taken from [13]. Is it necessary to talk about this in Part 2 of this standard?

6.2 PNML Documents for particular Petri Net Types

Based on the PNML Core Model of Clause 5.2, Clauses 5.3.1 and 5.3.2 define the concepts of two particular Petri net types, which restrict PNML Documents to the particular labels defined in the corresponding packages. 6.2.0.4

These packages define the labels that can or must be used in the particular Petri net. Here, we will show how to map such a package to the corresponding XML syntax. This mapping is the same for all type definitions. We will explain this mapping by the help of the example of Place/Transition Nets (see Fig. 3 and the PNML Document in Listing 1). 6.2.0.5

The PT-Net package defines three kinds of labels that can be used in a Place/Transition Net: Names, PTMarkings, and PTAnnotations. Every Net and each Node must have one Annotation Name, each Place must have one Annotation PTMarking, and each Arc must have one Annotation PTAnnotation. This is indicated by the compositions in the UML diagram. 6.2.0.6

The XML syntax for these labels is derived from the roles of these compositions. For example, an Annotation Name is mapped to an XML element `<name>`, an Annotation PTMarking is mapped to an XML element `<initialMarking>`, and an Annotation PTAnnotation will be mapped to an element `<inscription>` (see Listing 1). 6.2.0.7

Since all Labels in this package are derived from Annotation, all graphical elements defined for Annotations may occur as children in these elements. 6.2.0.8

In the PT-System package each label is defined to have a `<text>` element, which defines the 6.2.0.9

Listing 1: PNML code of the example net in Fig. 5

```
<pnml xmlns="http://www.example.org/pnml">
  <net id="n1" type="http://www.example.org/pnml/PTNet">
   <page id="top-level">
    <name>
      <text>An example P/T-net</text>
    </name>
    <place id="p1">
      <graphics>
        <position x="20" y="20"/>
      </graphics>
      <name>
        <text>ready</text>
        <graphics>
          <offset x="-10" y="-8"/>
        </graphics>
      </name>
      <initialMarking>
        <text>3</text>
        <toolspecific tool="PN4all" version="0.1">
          <tokenposition x="-2" y="-2" />
          <tokenposition x="2" y="0" />
          <tokenposition x="-2" y="2" />
        </toolspecific>
      </initialMarking>
    </place>
    <transition id="t1">
      <graphics>
        <position x="60" y="20"/>
      </graphics>
    </transition>
    <arc id="a1" source="p1" target="t1">
      <graphics>
        <position x="30" y="5"/>
        <position x="60" y="5"/>
      </graphics>
      <inscription>
        <text>2</text>
        <graphics>
          <offset x="15" y="-2"/>
        </graphics>
      </inscription>
    </arc>
   </page>
  </net>
</pnml>
```

actual content of this Annotation. For Place/Transition Net we do not need the structured element.

The corresponding classes define the XML content of the <text> element. 6.2.0.10

Editor's note 6.11: This should give the basic idea of this mapping. The same principle works for other type packages.

I will finish this later.

6.3 External Definitions (XML, XMLSchema, and MathML)

Editor's note 6.12: Here we must define the syntax of all classes prefixed with XML::, XMLSchema::, MathML::

For XML and XMLSchema, this is obvious. For MathML expressions this is more involved.

Editor's note 6.13: Some open issues:

- When combining different packages for different types, there might be name clashes. These name clashes could be resolved by using namespaces. Since this does not occur with the packages defined in Part 2, we did not mention this. But, there should be at least a comment, how this problem will be solved.

A RELAX NG Grammar for PNML (normative)

This Annex gives a complete definition of the PNML Core Model and its XML format in terms of a RELAX NG grammar. We start with a definition of basic PNML (i.e. PNML without pages) and then give the extensions for structured PNML (i.e PNML with pages).

Note that it is not the RELAX NG grammar that is defined here. It is the legal PNML Documents defined by this grammar that is normative.

Note that some syntactical restrictions of the PNML Core Model cannot be expressed in RELAX NG. Still, these restrictions expressed in the PNML Core Model are mandatory for valid PNML Documents.

Editor's note A.1: The RELAX NG grammars are adopted from an earlier version. They do not yet reflect the changes proposed at the Helsinki meeting (top-level page). Actually, there should be single a grammar, since we do not distinguish basic PNML Documents anymore.

I need support from Michael Weber for doing that.

Editor's note A.2: If there is a proposal in some other technology that precisely defines the mapping of UML Meta Models (packages) to XML, I would be happy to use that technology.

I included the RELAX NG grammar because it is the only technical definition of PNML Documents that we have right now.

A.1 RELAX NG Grammar for Basic PNML (without pages)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <a:documentation>
    Petri Net Markup Language schema
    RELAX NG implementation of basic PNML
    version: 1.3.2b
    according to the paper by Billington et al
    (c) 2001-2004
        Michael Weber (mweber@informatik.hu-berlin.de),
        Ekkart Kindler,
        Christian Stehno (for the graphical elements)
  </a:documentation>

  <start>
    <ref name="pnml.element"/>
  </start>

  <define name="pnml.element">
    <element name="pnml">
      <a:documentation>
        A PNML document consists of one or more Petri nets.
      </a:documentation>
      <oneOrMore>
        <ref name="pnml.content"/>
      </oneOrMore>
    </element>
  </define>
```

```
<define name="pnml.content">
  <ref name="net.element"/>
</define>

<define name="net.element">
  <element name="net">
    <a:documentation>
      A net has a unique identifier (id) and refers to
      its Petri Net Type Definition (PNTD) (type).
    </a:documentation>
    <attribute name="id">
      <data type="ID"/>
    </attribute>
    <attribute name="type">
      <ref name="nettype.uri"/>
    </attribute>
    <a:documentation>
      The sub-elements of a net may occur in any order.
      A net consists of several net labels (net.labels), several
      objects (net.content), tools specific information, and a set of
      graphical information in any order.
    </a:documentation>
    <interleave>
      <ref name="net.labels"/>
      <zeroOrMore>
        <ref name="net.content"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="toolspecific.element"/>
      </zeroOrMore>
      <optional>
        <element name="graphics">
          <ref name="netgraphics.content"/>
        </element>
      </optional>
    </interleave>
  </element>
</define>

<define name="nettype.uri">
  <a:documentation>
    The net type (nettype.uri) of a net should be redefined in a PNTD.
  </a:documentation>
  <data type="anyURI"/>
</define>

<define name="net.labels">
  <a:documentation>
```

```
      A net may have unspecified many labels. This pattern should be used
      within a PNTD to define the net labels.
    </a:documentation>
    <empty/>
  </define>

  <define name="net.content">
    <a:documentation>
      A net object is either a place, or a transition, or an arc.
    </a:documentation>
    <choice>
      <element name="place">
        <ref name="place.content"/>
      </element>
      <element name="transition">
        <ref name="transition.content"/>
      </element>
      <element name="arc">
        <ref name="arc.content"/>
      </element>
    </choice>
  </define>

  <define name="place.content">
    <a:documentation>
      A place may have several labels (place.labels) and the same content
      as a node.
    </a:documentation>
    <interleave>
      <ref name="place.labels"/>
      <ref name="node.content"/>
    </interleave>
  </define>

  <define name="place.labels">
    <a:documentation>
      A place may have unspecified many labels. This pattern should be used
      within a PNTD to define the place labels.
    </a:documentation>
    <empty/>
  </define>

  <define name="transition.content">
    <a:documentation>
      A transition may have several labels (transition.labels) and the same
      content as a node.
    </a:documentation>
    <interleave>
      <ref name="transition.labels"/>
      <ref name="node.content"/>
    </interleave>
  </define>

  <define name="transition.labels">
    <a:documentation>
      A transition may have unspecified many labels. This pattern should be
      used within a PNTD to define the transition labels.
    </a:documentation>
    <empty/>
  </define>

  <define name="node.content">
    <a:documentation>
      A node has a unique identifier.
    </a:documentation>
    <attribute name="id">
      <data type="ID"/>
    </attribute>
    <interleave>
      <a:documentation>
        The sub-elements of a node occur in any order.
        A node may consist of grahical and tool specific information.
      </a:documentation>
      <optional>
        <element name="graphics">
          <ref name="nodegraphics.content"/>
        </element>
      </optional>
      <zeroOrMore>
        <ref name="toolspecific.element"/>
      </zeroOrMore>
    </interleave>
  </define>

  <define name="arc.content">
    <a:documentation>
      An arc has a unique identifier (id) and
      refers both to the node's id of its source and
      the node's id of its target.
      In general, if the source attribute refers to a place,
      then the target attribute refers to a transition and vice versa.
    </a:documentation>
    <attribute name="id">
      <data type="ID"/>
    </attribute>
    <attribute name="source">
      <data type="IDREF"/>
    </attribute>
    <attribute name="target">
      <data type="IDREF"/>
    </attribute>
    <a:documentation>
      The sub-elements of an arc may occur in any order.
      An arc may have several labels. Furthermore, an arc may consist of
      grahical and tool specific information.
    </a:documentation>
    <interleave>
      <ref name="arc.labels"/>
      <optional>
        <element name="graphics">
          <ref name="edgegraphics.content"/>
        </element>
      </optional>
      <zeroOrMore>
        <ref name="toolspecific.element"/>
      </zeroOrMore>
    </interleave>
  </define>

  <define name="arc.labels">
    <a:documentation>
      An arc may have unspecified many labels. This pattern should be used
      within a PNTD to define the arc labels.
    </a:documentation>
    <empty/>
  </define>

  <define name="netgraphics.content">
    <a:documentation>
      Currently, there is no content of the graphics element of net defined.
    </a:documentation>
    <empty/>
  </define>

  <define name="nodegraphics.content">
    <a:documentation>
      The sub-elements of a node's graphical part occur in any order.
      At least, there must be exactly one position element.
      Furthermore, there may be a dimension, a fill, and a line element.
    </a:documentation>
    <interleave>
      <ref name="position.element"/>
      <optional>
        <ref name="dimension.element"/>
      </optional>
      <optional>
        <ref name="fill.element"/>
      </optional>
      <optional>
        <ref name="line.element"/>
      </optional>
    </interleave>
  </define>

  <define name="edgegraphics.content">
    <a:documentation>
      The sub-elements of an arc's graphical part occur in any order.
      There may be zero or more position elements.
      Furthermore, there may be a fill and a line element.
    </a:documentation>
    <interleave>
      <zeroOrMore>
        <ref name="position.element"/>
      </zeroOrMore>
      <!--
        <optional>
        <ref name="fill.element"/>
        </optional>
      -->
      <optional>
        <ref name="line.element"/>
      </optional>
    </interleave>
  </define>

  <define name="annotationgraphics.content">
    <a:documentation>
      An annotation's graphics part requires an offset element describing
      the offset the lower left point of the surrounding text box has to
      the reference point of the net object on which the annotation occurs.
      Furthermore, an annotation's graphic element may have a fill, a line,
      and font element.
    </a:documentation>
    <ref name="offset.element"/>
    <optional>
      <ref name="fill.element"/>
    </optional>
    <optional>
      <ref name="line.element"/>
    </optional>
    <optional>
      <ref name="font.element"/>
    </optional>
  </define>

  <define name="position.element">
```

```
    <a:documentation>
      A position element describes a Cartesian coordinate.
    </a:documentation>
    <element name="position">
      <ref name="coordinate.attributes"/>
    </element>
  </define>

  <define name="offset.element">
    <a:documentation>
      An offset element describes a Cartesian coordinate.
    </a:documentation>
    <element name="offset">
      <ref name="coordinate.attributes"/>
    </element>
  </define>

  <define name="coordinate.attributes">
    <a:documentation>
      The coordinates are decimal numbers and refer to an appropriate
      xy-system where the x-axis runs from left to right and the y-axis
      from top to bottom.
    </a:documentation>
    <attribute name="x">
      <data type="decimal"/>
    </attribute>
    <attribute name="y">
      <data type="decimal"/>
    </attribute>
  </define>

  <define name="dimension.element">
    <a:documentation>
      A dimension element describes the width (x coordinate) and height
      (y coordinate) of a node.
      The coordinates are actually positive decimals.
    </a:documentation>
    <element name="dimension">
      <attribute name="x">
        <data type="decimal"/>
      </attribute>
      <attribute name="y">
        <data type="decimal"/>
      </attribute>
    </element>
  </define>

  <define name="fill.element">
    <a:documentation>
```

```
      A fill element describes the interior colour, the gradient colour,
      and the gradient rotation between the colours of an object.  If an
      image is available the other attributes are ignored.
    </a:documentation>
    <element name="fill">
      <optional>
        <attribute name="color">
          <ref name="color.type"/>
        </attribute>
      </optional>
      <optional>
        <attribute name="gradient-color">
          <ref name="color.type"/>
        </attribute>
      </optional>
      <optional>
        <attribute name="gradient-rotation">
          <choice>
            <value>vertical</value>
            <value>horizontal</value>
            <value>diagonal</value>
          </choice>
        </attribute>
      </optional>
      <optional>
        <attribute name="image">
          <data type="anyURI"/>
        </attribute>
      </optional>
    </element>
  </define>

  <define name="line.element">
    <a:documentation>
      A line element describes the shape, the colour, the width, and the
      style of an object.
    </a:documentation>
    <element name="line">
      <optional>
        <attribute name="shape">
          <choice>
            <value>line</value>
            <value>curve</value>
          </choice>
        </attribute>
      </optional>
      <optional>
        <attribute name="color">
          <ref name="color.type"/>
        </attribute>
      </optional>
      <optional>
        <attribute name="width">
          <data type="decimal"/> <!-- actually, positive decimal -->
        </attribute>
      </optional>
      <optional>
        <attribute name="style">
          <choice>
            <value>solid</value>
            <value>dash</value>
            <value>dot</value>
          </choice>
        </attribute>
      </optional>
    </element>
  </define>

  <define name="color.type">
    <a:documentation>
      This describes the type of a color attribute. Actually, this comes
      from the CSS2 data type system.
    </a:documentation>
    <text/>
  </define>

  <define name="font.element">
    <a:documentation>
      A font element describes several font attributes, the decoration,
      the alignment, and the rotation angle of an annotation's text.
      The font attributes (family, style, weight, size) should be conform
      to the CSS2 data type system.
    </a:documentation>
    <element name="font">
      <optional>
        <attribute name="family">
          <text/>  <!-- actually, CSS2-font-family -->
        </attribute>
      </optional>
      <optional>
        <attribute name="style">
          <text/>  <!-- actually, CSS2-font-style -->
        </attribute>
      </optional>
      <optional>
        <attribute name="weight">
          <text/>  <!-- actually, CSS2-font-weight -->
        </attribute>
      </optional>
      <optional>
        <attribute name="size">
          <text/>  <!-- actually, CSS2-font-size -->
        </attribute>
      </optional>
      <optional>
        <attribute name="decoration">
          <choice>
            <value>underline</value>
            <value>overline</value>
            <value>line-through</value>
          </choice>
        </attribute>
      </optional>
      <optional>
        <attribute name="align">
          <choice>
            <value>left</value>
            <value>center</value>
            <value>right</value>
          </choice>
        </attribute>
      </optional>
      <optional>
        <attribute name="rotation">
          <data type="decimal"/>
        </attribute>
      </optional>
    </element>
  </define>

  <define name="toolspecific.element">
    <a:documentation>
      The tool specific information refers to a tool and its version.
      The further substructure is up to the tool.
    </a:documentation>
    <element name="toolspecific">
      <attribute name="tool">
        <text/>
      </attribute>
      <attribute name="version">
        <text/>
      </attribute>
      <ref name="anyElement"/>
    </element>
  </define>

  <define name="anyElement">
```

```
    <element>
      <anyName>
        <except>
          <nsName/>
        </except>
      </anyName>
      <zeroOrMore>
        <choice>
          <attribute>
            <anyName/>
          </attribute>
          <text/>
          <ref name="anyElement"/>
        </choice>
      </zeroOrMore>
    </element>
  </define>
</grammar>
```

A.2 RELAX NG Grammar for Structured PNML (with pages)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <a:documentation>
    Petri Net Markup Language schema
    RELAX NG implementation of structured PNML
    version: 1.3.2c
    according to the paper by Billington et al
    (c) 2001-2004
        Michael Weber, mweber@informatik.hu-berlin.de
        Ekkart Kindler
  </a:documentation>

  <include href="basicPNML.rng"/>

  <define name="net.content" combine="choice">
    <a:documentation>
      Now, a net object is additionally a page, a reference place, or
      a reference transition.
    </a:documentation>
    <choice>
      <element name="page">
        <ref name="page.content"/>
      </element>
      <element name="referencePlace">
```

```
        <ref name="refplace.content"/>
      </element>
      <element name="referenceTransition">
        <ref name="reftrans.content"/>
      </element>
    </choice>
  </define>

  <define name="page.content">
    <a:documentation>
      A page has a unique identifier (id).  It consists of several objects
      (the same as for a net), tool specific information, and graphical
      information.
    </a:documentation>
    <attribute name="id">
      <data type="ID"/>
    </attribute>
    <interleave>
      <zeroOrMore>
        <ref name="net.content"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="toolspecific.element"/>
      </zeroOrMore>
      <optional>
        <element name="graphics">
          <ref name="pagegraphics.content"/>
        </element>
      </optional>
    </interleave>
  </define>

  <define name="reference">
    <a:documentation>
      Here, we define the attribute ref including its data type.
      Modular PNML will extend this definition in order to change
      the behavior of references to export nodes of module instances.
    </a:documentation>
    <attribute name="ref">
      <data type="IDREF"/>
    </attribute>
  </define>

  <define name="refplace.content">
    <a:documentation>
      A reference place is a reference node.
    </a:documentation>
    <a:documentation>
      Validating instruction:
```

```
      - _ref_ MUST refer to _id_ of a reference place or of a place.
      - _ref_ MUST NOT refer to _id_ of its reference place element.
      - _ref_ MUST NOT refer to a cycle of reference places.
    </a:documentation>
    <ref name="refnode.content"/>
  </define>

  <define name="reftrans.content">
    <a:documentation>
      A reference transition is a reference node.
    </a:documentation>
    <a:documentation>
      Validating instruction:
      - The reference (ref) MUST refer to a reference transition or to a
        transition.
      - The reference (ref) MUST NOT refer to the identifier (id) of its
        reference transition element.
      - The reference (ref) MUST NOT refer to a cycle of reference transitions.
    </a:documentation>
    <ref name="refnode.content"/>
  </define>

  <define name="refnode.content">
    <a:documentation>
      A reference node has the same content as a node.
      It adds a reference (ref) to a (reference) node.
    </a:documentation>
    <ref name="node.content"/>
    <ref name="reference"/>
  </define>

  <define name="pagegraphics.content">
    <a:documentation>
      Currently, there is no content of the graphics element of page defined.
    </a:documentation>
    <empty/>
  </define>
</grammar>
```

Editor's note A.3: It appears that pages do not have names according to this definition. I think, we agreed that pages could have names. This must be checked.

Editor's note A.4: There is a TeXnical Problem with italics in comments in combination with quotes. This needs some polishing.

B RELAX NG Grammars for special types (normative)

B.1 Place/Transition Nets

B.1.1 The labels for P/T Nets

First, there is a definition of the labels of P/T Nets

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE grammar PUBLIC "-//thaiopensource//DTD RNG 20010705//EN" "">

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

 <a:documentation>
    Conventions Document (conv.rng)
    RELAX NG implementation
    version: 0.1 (2003-06-18)
    (c) 2003
        Michael Weber, mweber@informatik.hu-berlin.de
 </a:documentation>

 <a:documentation>
    First, we define several short cuts for label definitions.  They are
    used for simple data or if the label data are not really specified.
    The usage of these short cuts is documented at the end of this file.
    Furthermore, these short cuts also can be used if the rest of the
    Conventions Document is ignored.
 </a:documentation>

 <define name="attribute.content">
  <a:documentation>
     The definition attribute.content describes the content of a
     simple text label without graphics (i.e. attribute to net objects).
     It can be used as a schema for those labels.
  </a:documentation>
  <optional>
   <element name="text">
    <a:documentation>
       A text label may have a value;
       if not, then there must be a default.
    </a:documentation>
    <text/>
   </element>
  </optional>
 </define>

 <define name="annotationstandard.content">
```

```
  <a:documentation>
     The definition annotationstandard.content describes the
     standard stuff of an annotation.
     Each annotation may have graphical or tool specific information.
  </a:documentation>
  <optional>
   <element name="graphics">
    <ref name="annotationgraphics.content"/>
   </element>
  </optional>
  <zeroOrMore>
   <ref name="toolspecific.element"/>
  </zeroOrMore>
 </define>

 <define name="simpletextlabel.content">
  <a:documentation>
     A simple text label is an annotation to a net object containing
     unspecified text.
     Its sub-elements occur in any order.
     A simple text label behaves like an attribute to a net object.
     Furthermore, it contains the standard annotation content.
  </a:documentation>
  <interleave>
   <ref name="attribute.content"/>
   <ref name="annotationstandard.content"/>
  </interleave>
 </define>

 <define name="nonnegativeintegerlabel.content">
  <a:documentation>
     A non negative integer label is an annotation with a natural
     number as its value.
     Its sub-elements occur in any order.
     It contains the standard annotation content.
  </a:documentation>
  <interleave>
   <element name="text">
    <data type="nonNegativeInteger"
          datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
   </element>
   <ref name="annotationstandard.content"/>
  </interleave>
 </define>

 <define name="complexlabel.content">
  <a:documentation>
     A complex label is a sub-structured annotation to a net object.
     The definition complexlabel.content can be used as a general
```

```
      schema for those labels.
      A complex label is at least a simple text label. It may further
      contain XML structured data.  The subelement text contains the
      string representation of the structured data.
   </a:documentation>
   <interleave>
    <ref name="attribute.content"/>
    <optional>
     <element name="structure">
      <ref name="anyElement"/>
     </element>
    </optional>
    <ref name="annotationstandard.content"/>
   </interleave>
  </define>

<!-- The following definitions are the Conventions Document’s
      label definitions -->

  <define name="Name">
   <a:documentation>
      Label definition for a user given identifier of an element describing
      its meaning.
      <contributed>Michael Weber</contributed>
      <date>2003-06-16</date>
   </a:documentation>
   <element name="name">
    <ref name="simpletextlabel.content"/>
   </element>
  </define>

  <define name="PTMarking">
   <a:documentation>
      Label definition for initial marking in nets like P/T-nets.
      <contributed>Michael Weber</contributed>
      <date>2003-06-16</date>
      <reference>
       W. Reisig: Place/transition systems. In: LNCS 254. 1987.
      </reference>
   </a:documentation>
   <element name="initialMarking">
    <ref name="nonnegativeintegerlabel.content"/>
   </element>
  </define>


  <define name="PTArcInscription">
   <a:documentation>
      Label definition for arc inscriptions in nets like P/T-nets.
      <contributed>Michael Weber</contributed>
      <date>2003-06-16</date>
      <reference>
       W. Reisig: Place/transition systems. In: LNCS 254. 1987.
      </reference>
   </a:documentation>
   <element name="inscription">
    <ref name="nonnegativeintegerlabel.content"/>
   </element>
  </define>

</grammar>
```

B.1.2 The Grammar for P/T Nets

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar ns="http://www.informatik.hu-berlin.de/top/pnml/ptNetb"
         xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

 <a:documentation>
    Petri Net Type Definition for Place/Transition nets (bases on basic PNML)
    RELAX NG implementation of ptNetb.pntd
    version: 1.0
    (c) 2001-2003
        Michael Weber, mweber@informatik.hu-berlin.de
 </a:documentation>

 <a:documentation>
    We include PNML with the correct URI for our Petri Net Type Definition.
 </a:documentation>
 <include href="http://www.informatik.hu-berlin.de/top/pnml/basicPNML.rng">
  <define name="nettype.uri" combine="choice">
   <a:documentation>
      We define the net type URI declaring the namespace of this
      Petri net type definition.
   </a:documentation>
   <value>http://www.informatik.hu-berlin.de/top/pntd/ptNetb</value>
  </define>
 </include>

 <a:documentation>
    All labels of this Petri net type come from the Conventions Document.
 </a:documentation>
 <include href="http://www.informatik.hu-berlin.de/top/pnml/conv.rng"/>

 <define name="net.labels" combine="interleave">
  <a:documentation>
```

```
     A P/T net may have a name.
  </a:documentation>
  <optional><ref name="Name"/></optional>
 </define>

 <define name="place.labels" combine="interleave">
  <a:documentation>
     A place of a P/T net may have a name and an initial marking.
  </a:documentation>
  <interleave>
   <optional><ref name="PTMarking"/></optional>
   <optional><ref name="Name"/></optional>
  </interleave>
 </define>

 <define name="transition.labels" combine="interleave">
  <a:documentation>
     A transition of a P/T net may have a name.
  </a:documentation>
  <optional><ref name="Name"/></optional>
 </define>

 <define name="arc.labels" combine="interleave">
  <a:documentation>
     An arc of a P/T net may have an inscription.
  </a:documentation>
  <optional><ref name="PTArcInscription"/></optional>
 </define>

</grammar>
```

Editor's note B.1: There seems to be a slight difference between the Type Model of Place/Transition Nets as defined in Clause 5.3.1. For example, there can be Arc Annotations with value 0 according to the RELAX NG PNTD, which is not possible in the Meta Model.

This needs to be checked and corrected.

B.2 HLPNGs

Editor's note B.2: This is not yet the RELAX NG grammar for HLPNGs! The reason is that we did not work out the details on how to include MathML (in RELAX NG or any other technology).

C Transformation to SVG (non-normative)

Editor's note C.1: In order to define the graphical appearance of a PNML Document, this Annex introduces transformation to SVG. Whether this should be part of the Standard (normative or non-normative) needs to be discussed.

The text here was taken from an earlier version of this Standard.

In order to give a precise definition of the graphical presentation of a PNML document with all its graphical features, we define a translation to SVG. Petri net tools that support PNML can visualize Petri nets using other means than SVG, but the SVG translation can act as a reference model for such visualizations. Technically, this translation is done by means of an XSLT stylesheet. The basic idea of this transformation was already presented in. A complete XSLT stylesheet can be found on the PNML web pages. C.0.0.1

Editor's note C.2: The transformations still need some polishing; then they could go to the Appendix (normative or non-normative)

Transformations of basic PNML. The overall idea of the translation from PNML to SVG is to transform each PNML object to some SVG object, where the attributes of the PNML element and its child elements are used to give the SVG element the intended graphical appearance. C.0.0.2

As expected, a place is transformed into an ellipse, while a transition is transformed into a rectangle. Their position and size are calculated from the <position> and <dimension> elements. Likewise, the other graphical attributes from <fill> and <line> can be easily transformed to the corresponding SVG attributes. C.0.0.3

An annotation is transformed to SVG text such as `name: someplace`. The location of this text is automatically computed from the attributes in <offset> and the position of the corresponding object. For an arc, this reference position is the center of the first line segment. If there is no <offset> element, the transformation uses some default value, while trying to avoid overlapping. C.0.0.4

An arc is transformed into a SVG path from the source node to the target node – possibly via some intermediate points – with the corresponding attributes for its shape. The start and end points of a path may be decorated with some graphical object corresponding to the nature of the arc (e.g. inhibitor). The standard transformation supports arrow heads as decorations at the end, only. The arrow head (or another decoration) should be exactly aligned with the corresponding node. This requires computations using complex operations that are neither available in XSLT nor in SVG – the current transformation uses recursive approximation instead. C.0.0.5

Transformations for structured PNML. Different pages of a net should be written to different SVG files since SVG does not support multi-image files. Unfortunately, XSLT does not support output to different files yet, but XSLT 2.0 will. Hence, a transformation of structured PNML to SVG will be supported once XSLT 2.0 is available. C.0.0.6

The transformations for reference places and reference transitions are similar to those for places and transitions. In order to distinguish reference nodes from other nodes, reference nodes are drawn slightly translucent and an additional label gives the name of the referenced object. C.0.0.7

Editor's note C.3: A concrete XSLT transformation could be contributed by Christian Stehno.

D API Framework for Petri Net type Meta-models (non-normative)

Editor's note D.1: This is a verbatim copy of an AFNOR proposal; there was no time adapting its layout, and spelling etc. to the standard. This will be polished in the next version.

D.1 Introduction

A software framework is proposed to tools developers along with this International Standard with the following purposes :

- **Making this International Standard applicable**. Indeed, PNML transfer format has been designed to enable easy interchange of many variants of Petri nets between Petri net tools. Therefore, interoperability will be made possible by compatibility between the main Petri nets types and by their extensibility to other variants.
- **Easing this International Standard applicability** by providing tools designers with immediate low-cost integration of PNML to their applications and thus making their convergence to this Standard homogeneously through the proposed framework. Such an approach will ensure tools compatibility with this Standard without having to perform the tedious and costly work of dealing with whatever version of PNML. In fact, designing a program to load/save Petri net models according to the Standard could be an error-prone task and may lead to compliance problems.
- **Providing tools developers with standard APIs** to manipulate Petri net models. These APIs are automatically generated from the corresponding Petri net type meta-model. They also have to be multi-platform.
- **Enabling Petri nets tools designers to add their own extensions** to the main Petri nets types defined by this Standard while keeping some level of conformance. This is achieved by providing tools designers with guidelines to make these extensions. Consequently the generated APIs will adapt to these extensions.

The proposed object-oriented framework has been built by means of model engineering techniques, thanks to OMG's Model Driven Architecture (MDA) approach guidelines [**?**] and related technologies such as MOF [**?**] and its implementation through Eclipse Modelling Framework [**?**].

In the following, we first describe the methodology to build the standard APIs from each main Petri net type meta-model, then we show an example API usage in the example context of a specific Petri net modelling tool such as CPN-AMI [**?**]. This step helps taking a tool developer point of view, regarding the API use, in order to estimate the work required from him/her. Finally, we provide the main guidelines to build extensions to the Petri net type meta-models and show an example.

D.2 Methodology

D.2.1 Overview

The API framework has been designed following a three-level structuring approach:

1. First of all, a *Petri Net Core Metamodel* (PNCM) has been created. It is based on PNML Core Model and Place/Transition System Type Model defined by this Standard. Thus, it is *minimal* for P/T nets. This metamodel represents the modelling process starting point. It means that from the PNCM, a hierarchical classification allows for structuring the main Petri Net type Models defined by this Standard and through its future versions. Each main Petri Net Type Model represents a significant Petri net family (e.g, P/T Systems, High-Level Petri Nets, etc.) which could contain many variants (e.g., P/T Systems with capacity, Timed Petri nets with priorities, etc.), thus meeting the extensibility goal. From this Standard point of view, the classification approach roughly corresponds to the relationship between the Petri Net type Metamodels packages, symbolically represented by the UML *merge* concept.

2. From a given Petri Net Type Metamodel, *a tailored API is automatically generated* to manipulate models structured by the given metamodel specification. Another programming interface is generated to import/export the models from/into PNML documents, according to the specified Petri Net Type Metamodel and its variants. This import/export API uses the tailored one.

3. To start model manipulation (i.e., creation, modification, navigation to fetch information) and having them eventually written in PNML or translated into a tool internal Petri net representation, a client program is needed. Writing this client program should be done by tools developers in order to perform the mapping between their own tool internal representation and the information retrieved or injected into the models. At this stage, the only requirement from tools developers is to know the specified Petri Net Type Metamodel elements and their structuring relationships, in order to use the corresponding API to create models and fetch information.

Figure 6 illustrates the three-level approach described above.

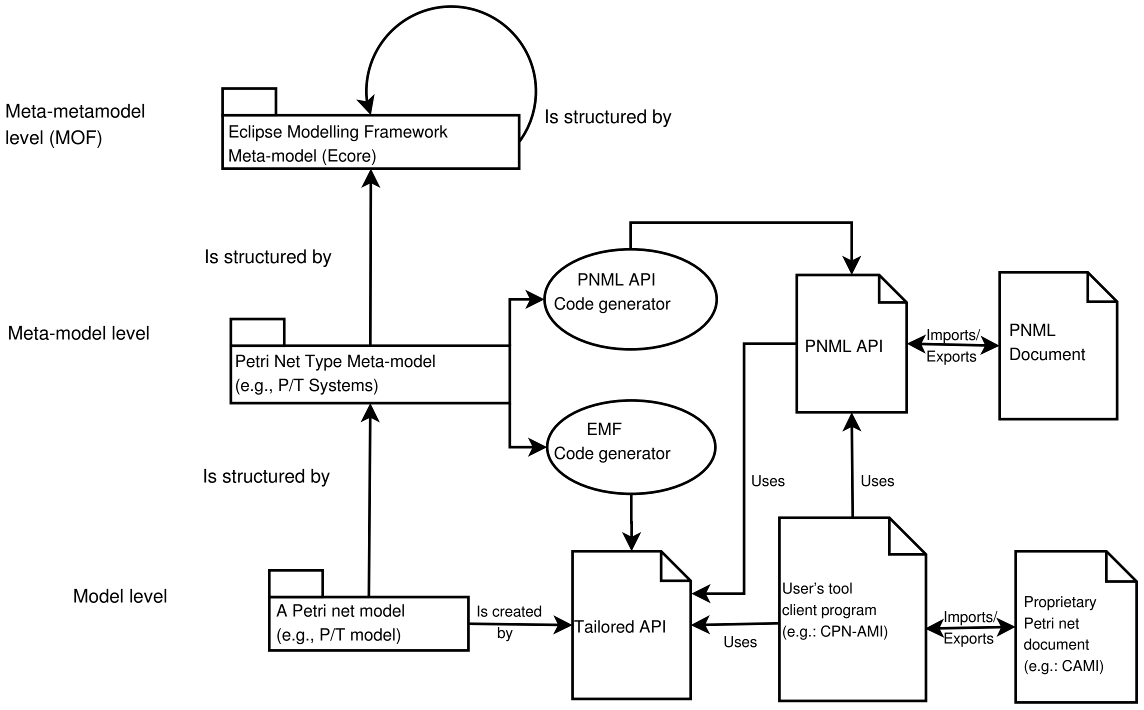

Meta-metamodel level (MOF)
Eclipse Modelling Framework Meta-model (Ecore)
Is structured by
Is structured by
Meta-model level
Petri Net Type Meta-model (e.g., P/T Systems)
PNML API Code generator
EMF Code generator
PNML API
Imports/ Exports
PNML Document
Is structured by
Uses
Uses
Model level
A Petri net model (e.g., P/T model)
Is created by
Tailored API
Uses
User's tool client program (e.g.: CPN-AMI)
Imports/ Exports
Proprietary Petri net document (e.g.: CAMI)

Figure 6: The production chain

In the following, the main Petri net Type Meta-models defined in this Standard will be introduced, along with their designing process.

D.2.2 P/T System Type Meta-model

In this section,the P/T Systems Type meta-model is specified. Since it is an implementation of the corresponding formal Type meta-model introduced in this Standard in clause 5.3, its designing process is explained. Especially, when speaking of involved technologies such as Eclipse Modelling Framework, model productivity is considered and consequently restrictions are set on the formal Type meta-model, without any lost of requirements.

Note: More details to follow

D.2.3 HLPNG Type Meta-model

In this section,the HPLNGs Type meta-model is specified. The designing approach is the same as for P/T Systems. Relationship between these two meta-models, when considering the UML formal *merge* concept, will be explained. Some consideration about MathML will be introduced.

Note: More details to follow

D.3 Using the API framework

Code generation is the next step to the meta-models specification. For each meta-model, two application programming interfaces (API) and their implementation are generated in Java programming language. Their main functions are shown, along with their use. As said before in the introductory section of this annex, a tool developer point of view is adopted in order to evaluate the low-cost integration of this Standard to an existing tool. CPN-AMI [**?**] and his internal proprietary CAMI format will be taken as an example.

Note: More details to follow

D.4 Guidelines to extending Petri Net Type Meta-models

There are many variants of Petri nets, when considering each main Petri Net Type. This Standard has defined some of them, namely, P/T Systems and HLPNGs. However, tools developers may need using these variants (e.g., P/T Systems with capacity, etc.) even if they are not yet formally defined. Therefore, it would be interesting to have means to extend the provided meta-models by the proposed framework in order to adapt the corresponding APIs while keeping some level of conformance to the Standard. To meet this expectation somehow, guidelines are provided to help tools developers to enhance the given meta-models. Following mostly these guidelines ensures that the generated API will adapt to the extended meta-models.

Note: More details to follow

Note: More details to follow

Bibliography

[1] R. Bastide, J. Billington, E. Kindler, F. Kordon, and K. H. Mortensen, editors. *Meeting on XML/SGML based Interchange Formats for Petri Nets*, Århus, Denmark, June 2000. 21st ICATPN.

[2] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletter*, 49:9–27, October 1995.

[3] G. Berthelot, J. Vautherin, and G. Vidal-Naquet. A syntax for the description of Petri nets. *Petri Net Newsletter*, 29:4–15, April 1988.

[4] Jonathan Billington, Søren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, technology, and tools. In W. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003, 24*th *International Conference*, volume 2679 of *LNCS*, pages 483–505. Springer, June 2003.

[5] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs (ed.). Cascading Style Sheets, level 2 – CSS2 Specification. URL `http://www.w3.org/TR/CSS2`, 1998.

[6] J. Clark. TREX – tree regular expressions for XML. URL `http://www.thaiopensource.com/trex/`. 2001/01/20.

[7] J. Clark and M. Murata (eds.). RELAX NG specification. URL `http://www.oasis-open.org/committees/relax-ng/`. 2001/12/03.

[8] J. Clark (eds.). XSL Transformations (XSLT) Version 1.0. URL `http://www.w3.org/TR/XSLT/xslt.html`, 1999.

[9] J. Ferraiolo, F. Jun, and D. Jackson (eds.). Scalable Vector Graphics (SVG) 1.1 Specification. URL `http://www.w3.org/TR/SVG11/`, 2003.

[10] ISO/IEC/JTC1/SC7. Software Engineering - High-Level Petri Nets - Concepts, Definitions and Graphical Notation. ISO/IEC 15909-1, December 2004.

[11] M. Jüngel, E. Kindler, and M. Weber. The Petri Net Markup Language. *Petri Net Newsletter*, 59:24–29, 2000.

[12] M. Jüngel, E. Kindler, and M. Weber. The Petri Net Markup Language. In S. Philippi, editor, *7. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 47–52, Universität Koblenz-Landau, Germany, June 2000. AWPN. URL `http://www.informatik.hu-berlin.de/top/pnml/`.

[13] E. Kindler and M. Weber. A universal module concept for Petri nets. An implementation-oriented approach. Informatik-Berichte 150, Humboldt-Universität zu Berlin, June 2001.

[14] A. M. Koelmans. PNIF language definition. Technical report, Computing Science Department, University of Newcastle upon Tyne, UK, July 1995. version 2.2.

[15] F. Kordon and L. Petrucci. Structure of abstract syntax trees for coloured nets in PNML. version 0.2 (draft), June 2004.

[16] R. B. Lyngsø and T. Mailund. Textual interchange format for high-level Petri nets. In *Proc. Workshop on Practical use of Coloured Petri Nets and Design/CPN*, pages 47–63, Department of Computer Science, University ofÅrhus, Denmark, 1998. PB-532.

[17] T. Mailund and K. H. Mortensen. Separation of style and content with XML in an interchange format for high-level Petri nets. In Bastide et al. [1], pages 7–11.

[18] Petri Net Markup Language. URL `http://www.informatik.hu-berlin.de/top/pnml/`. 2001/07/19.

[19] M. Sperberg-McQueen and H. Thompson (eds.). XML Schema. URL `http://www.w3.org/XML/Schema`, April 2000. 2002-03-22.

[20] C. Stehno. Petri Net Markup Language: Implementation and Application. In J. Desel and M. Weske, editors, *Promise 2002*, volume P-21 of *Lecture Notes in Informatics*, pages 18–30. Gesellschaft für Informatik, 2002.

[21] O. Sy, M. Buffo, D. Buchs, F. Kordon, and R. Bastide. An experimental approach towards the XML representation of Petri net models. Technical Report 2000/336, École Polytechnique Fédéral de Lausanne, Departement D'Informatique, June 2000.

[22] M. Weber and E. Kindler. The Petri Net Markup Language. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Petri Net Technology for Communication Based Systems*, number 2472 in Lecture Notes in Computer Science, pages 124–144. Springer, Berlin Heidelberg, 2003.

[23] G. Wheeler. A textual syntax for describing Petri nets. Foresee design document, Telecom Australia Research Laboratories, 1993. version 2.

[24] World Wide Web Consortium (W3C) (ed.). Extensible Markup Language (XML). URL `http://www.w3.org/XML/`. 2000/10/06.

Editor's note D.2: These references are very prelimary. The strong bias on my (E.K.) own publications should be eliminated.