High-level Petri Nets – Transfer Format

– Working Draft of the International Standard ISO/IEC 15909 Part 2 –

Version 0.5.0

Ekkart Kindler
University of Paderborn

November 8, 2004

Contents

Preface

...

Editor's note: At the meeting in Brisbane (May 2004), there was a request to use a Word template in order to make this document conform to JTC1 format. The problem with this request is that we use much material from scientific papers, which are all written in LaTeX format. There is no easy or efficient way to convert LaTeX to Word (in particular concerning formating issues of XML and code examples). Moreover, working with Word on this kind of material requires extra effort, even after the material has been converted to Word (for example, there is no way to properly maintain a Word document under the control of a version management system).

Therefore, this document is still written in LaTeX. I hope that, eventually, we will have a LaTeX style, which makes this document appear in the required format.

If ISO/IEC insist on using Word for these working drafts, I (Ekkart Kindler) am asking for someone else to do the editor's job.

Introduction

0.0.0.1 This is the working draft (version 0.5) of ISO/IEC 15909-2 for a Transfer Format for High-level Petri Nets. It is based on [22, 4, 15] and comments from the ISO/IEC JTC1/SC7/WG19 meeting in Brisbane 10-14 May 2004.

0.0.0.2 At its meeting in Brisbane, WG19 agreed that the scope of Part 2 will be restricted to the scope of Part 1. Thus no modularity constructs will be included in this initial standard. The inclusion of modularity will occur in Part 3 of the standard, and then once this is done, the Transfer Format will be enhanced either by addenda or as part of the 5 year revision cycle.

There is no problem with including ideas for extensions, such as modularity and time, in an informative annex of Part 2. There is also no problem with commencing a draft of Part 3, if resources permit. It was agreed that publishing a first cut at the Transfer Format in a timely manner was much more important than trying to get a more complete version published, and not meeting our schedule. The work is currently delayed by one year from the schedule proposed in the NWI. A new schedule for Part 2 is as follows:

Due Date	Action
1 July 2004	Circulate WD to SC7 after discussion at Bologna Workshop
September 2004	Format according to JTC1 Word Template
October 2004	Complete P/T system syntax, revise Meta Model and its explanation
November 2004	Interim Meeting of WG19: Discuss and resolve National Body comments on WD
April 2005	Include HLPNG syntax and circulate to WG19
May 2005 Plenary	Incorporate WG19 comments, advance to CD
1 July 2005	Send to SC7 for CD registration and ballot
1 July 2006	Send to SC7 for FCD ballot
1 July 2007	Send to SC7/ITTF for FDIS ballot

Editor's note: I agree to the statement that the work on Part 2 should not be delayed by introducing concepts that were not covered in Part 1 of the International Standard. But, I feel that including modularity would not delay the work; actually, at the Bologna meeting there were many request to include modularity and there was aggreement on the concepts of modular PNML.

It is correct that Part 1 does not deal with pages and modules. But when it comes to transfer formats, we must deal with such concepts. Todays nets are much to large to fit on a single paper. Therefore, most tools support one concept or another for splitting a net up into different parts. If the transfer format does not support splitting a single net into parts, it will be completely useless. Even worse, every tool will come up with its own extension to deal with this problem, which will undermine the standard.

PNML's page concept was accepted by the Petri net community right away. Therefore, pages are included in this draft.

Even modular PNML got strong support at the Bologna meeting. Therefore, including this module concept should be seriously considered, because some tools have problems mapping there structuring concepts to the page concept alone.

0.0.0.3 National Bodies are asked to comment on this draft.

0.0.0.4 Informal comments can be sent directly to the editor (Ekkart Kindler, `kindler@upb.de`) or can be discussed on the PNX-Mailinglist (see `http://www.informatik.hu-berlin.de/top/PNX/` for details).

1 Scope

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

1.0.0.2 There are many different variants of Petri nets. High-level Petri Nets as defined in ISO/IEC 15909-1 are one version that attempts to cover many variants of Petri nets, although not all. This has been achieved by using the widely accepted coloured Petri net semantics as the semantic model, and an abstract mathematical syntax based on algebraic specification techniques. This covers Petri net classes such as Elementary nets, Place/Transition systems, well formed nets and many-sorted algebraic nets.

1.0.0.3 One objective of this transfer format 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, should be easily ignored without losing all information.

Editor's note: This is the main reason for introducing the concepts of the PNML Core Model, the Features Definition Interface, and the Petri Net Type Definition Interface first, which might appear a bit complicated on first glance. The reason for introducing it was a major requirement on an interchange format that was identified by the Petri net community (in Aarhus in June 2000): the format must support the exchange of Petri Nets or at least most parts of it, even if the tools do not support exactly the same type of Petri net.

Maybe, we should stress this even more in this section

1.0.0.4 Therefore, 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*. Based on this Core Model, this International Standard defines an interface for defining the special features of particular versions of Petri nets, the *Features Definition Interface*. Moreover, this International Standard defines an interface for choosing and combining features defined according to the Features Definition Interface in order to define a transfer format for a particular version of Petri nets. This is called the *Petri Net Type Definition Interface*. A definition of a *Petri Net Type* according to this interface is called a *Petri Net Type Definition* (PNTD).

1.0.0.5 In order to define a concrete transfer format for Petri Nets, this International Standard specifies also a Petri Net Type Definition for Place/-Transition Nets and for High-level Petri Nets as defined in Part 1 of this International Standard (ISO/IEC 15909-1). It thus defines a transfer format for High-level Petri Nets. Well-Formed Petri Nets are defined as a special case of High-level Petri Nets. 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 (ISO/IEC 15909-3).

1.0.0.6 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. These will be defined in ISO/IEC 15909-3.

Editor's note: Including modularity into Part 2 should be seriously considered. See editor's comment in the introduction.

2 Conformance

2.0.0.1 There are different kinds of Conformance to this International Standard.

2.0.0.2 First, documents can be conformant to the Features Definition Interface as defined in this International Standard. Such a document is called a Feature Definition.

2.0.0.3 Second, documents can be conformant to the Petri Net Type Definition Interface as defined in this International Standard. Such a document is called a Petri Net Type Definition (PNTD).

2.0.0.4 This International Standard itself defines three Petri Net Types: Place/Transition Nets, Well Formed Nets, and High-level Petri Net Graphs as defined in Part 1 of this International standard. These three Petri Net Types are called Standard Petri Net Types.

Editor's note: Right now, I am not sure whether we need separate Type Definitions for High-level Petri Net Graphs and Well-Formed Nets. I think that we could use the same Type Definition, with some additional restrictions on the built-in sorts and operations. This needs to be discussed.

Editor's note: There could be more Standard Petri Net Types. But, these three are the ones included right now in this International Standard. Perhaps, we should mention somewhere that more Standard Petri Net Types could be defined, and a procedure how to publish such Standard Petri Net Types. This could be done in Appendix D.

2.0.0.5 Third, a document can be conformant to the PNML Core Model, but not (necessarily) referring to a Petri Net Type. Such documents are called PNML documents.

2.0.0.6 Forth, a PNML document can be conformant to some Petri Net Type Definition. Such a document is called a PNML document of this particular Petri Net Type.

2.0.0.7 Fifth, a PNML document that are conformant to a Petri Net Type Definition, which is defined within this International Standard, are called PNML documents of this Standard Petri Net Type.

3 Normative References

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

Editor's note: Here is only a list of standards with their names, without the necessary details. These details need to be filled in later.

- ISO/IEC 15909-1 Software and system engineering – High-level Petri Nets, Concepts, definition and graphical notation.
- UML

Editor's note: Currently, we do not use special concepts of UML. Therefore, the precise version of UML should not matter. Do we refer to UML 1.4.1 (ISO/IEC 19501) or to UML 2.0?

There was a proposal to use XMI for making the meta model an XML transfer format. This is not yet captured here. In case we use XMI, it must be added here. When using XMI, the problem of exchanging Petri nets sharing some features but are not exactly of the same type is yet to be solved.

- XML
- XML Schema
- XSLT
- CSS
- SVG
- RELAX NG

4 Terms and Definitions

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

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

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

4.1.0.2 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, Petri Net, Node, Place/Transition Net, Operator, Place, Place Type, Signature, Sort, Argument Sort, Range Sort, Term, Closed Term (Ground Term), Transition, Transition Condition, Type.

4.2 Definitions for Part 2

4.2.0.1 Annotation A label represented as text near the corresponding object in a net graph.

4.2.0.2 Attribute A label that governs the form or shape of the corresponding object in a net graph.

4.2.0.3 Conventions Document A document containing Feature Definitions that are considered to be Standard Features. These Features can be used in Petri Net Type Definitions.

Editor's note: If this Standard will be confined to a transfer format for P/T-Systems and HLPNGs only, the Conventions Document could be omitted here. This Standard defines all mechanisms for defining Features and Petri Net Types anyway. So it does not hurt to introduce the concept of a Conventions Document.

4.2.0.4 Feature A Feature is a particular extension used in some version or several versions of Petri nets. A feature is represented by a label that may be or must be present at some arc or node of the net or as a label of the net itself. A feature definition defines the structure of the information within such a label. The same feature may occur in different variants of Petri nets. Therefore, there is a separate interface for defining features, which can then be used in Petri Net Type Definitions.

4.2.0.5 Features Definition The definition of a feature according to the Features Definition Interface. Basically, it defines a label that may or must be present at a node or arc of the net graph or in the net itself.

4.2.0.6 Features Definition Interface An interface used for defining a feature or a set of features. It defines the structure of a document containing Feature Definitions. Basically, it defines the structure of labels that may or must be present at an object of the net graph or in the net itself.

4.2.0.7 Standard Feature A feature that is defined in a (informal or formal) standard. It could be published in a Conventions Document.

4.2.0.8 Graphical Information The information required to define the graphics associated with each object of a net graph as well as its labels.

4.2.0.9 Global Label A label associated with the Petri net graph itself, rather than with a Petri net object.

4.2.0.10 Label Information associated with a net graph or one of its objects. The labels that may be used in a Net Graph will be defined in a Petri Net Type Definition.

4.2.0.11 High-level Petri Net graph with pages A high-level Petri net graph that is structured using pages.

4.2.0.12 Net Graph with Pages A net graph that is structured using pages.

4.2.0.13 Object (of a Net Graph) The arcs and nodes a net graph. In a net graph with pages, pages and reference nodes are also considered objects.

4.2.0.14 Page A structuring mechanism used to split large Petri net graphs into smaller parts. The Petri net will be split into parts called pages.

4.2.0.15 Petri Net Document A document that contains one or more Petri nets (of some type).

4.2.0.16 Petri Net Type Definition (PNTD) A document defining the format of a particular version of Petri nets. The structure of this document is defined by the Petri Net Type Definition Interface. A Petri Net Type Definition, can refer to Features defined in some Feature Definitions documents.

4.2.0.17 Petri Net Type Definition Interface There are many different versions and variants of a Petri net. Basically, all versions of Petri Net Graphs share the underlying net structure. The specifics of a particular version of Petri nets will be defined in terms of labels that may or must be attached to the objects of a Petri Net Graph. The Petri Net Type Definition Interface is a technique for defining the format of a particular version of Petri net. It defines the structure of a document defining a Petri net type.

4.2.0.18 Standard Petri Net Type Definition (PNTD) A Petri Net Type Definition that is published in this or other (formal or informal) standards. In this International Standard we define Standard Petri Net Types for Place/Transition-Systems, High-level Petri Net Graphs, and Well-Formed Nets.

4.2.0.19 PNML Core Model The model of the graph structure that is common to all versions of Petri nets.

4.2.0.20 Reference Node A representative of a node defined on another page of a HLPNG with pages. This node does not carry any semantical information itself. Rather, it has a pointer to the other node.

4.2.0.21 Reference Place A reference node that is a place. Its reference must point to another reference place or to a place.

4.2.0.22 Reference Transition A reference node that is a transition. Its reference must point to another reference transition or to a transition.

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

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

4.2.0.25 Tool Specific Information Information associated with net graph objects 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
- PNML: Petri Net Markup Language
- PNTD: Petri Net Type Definition
- UML: Unified Modelling Language
- XML: eXtensible Markup Language
- SVG: Scalable Vector Graphics
- XSLT: eXtensible Stylesheet Language Transformations
- CSS: Cascading Stylesheets

5 PNML Concepts

5.1 General Principles

5.1.0.1 This International Standard defines a transfer format for High-level Petri Net Graphs. This transfer format has been designed to be extended for other or 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 attached to the objects of this graph or attached to the graph itself. 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 discussed in Clause 5.2.

Editor's note: The term "PNML Meta Model" was replaced by the term "PNML Core Model", which is a consequence of comments and discussions at the Brisbane meeting and the Bologna meeting. Now, we have meta models for the concrete Petri Net Types, too. Therefore, we must clearly distinguish the part of the Meta Model belonging to PNML, which is called Core Model, and the part defined by a Petri Net Type Definition, which is called a Type Model (or Petri Net Type Model).

5.1.0.2 Note that the Core Model will be presented by the help of UML class diagrams in this Clause, which does not define the concrete XML syntax of for PNML documents. Clause 6 will define the mapping of the UML Meta Models to XML syntax by the help of RELAX NG technology.

5.1.0.3 The PNML Core Model imposes no restrictions on labels. Therefore, the Core Model can represent any kind of Petri net. Due to this generality of the Core Model, there can be even models that do not correspond to a Petri net at all. In order deal with this problem, PNML comes with a *Petri Net Type Definition Interface*, which supports the definition or Petri net types. Such a *Petri Net Type Definition* defines the instances which make up a Petri net of this particular kind. Basically, a Petri Net Type Definition defines the legal labels of this type. So the PNML Core Model in combination with a Petri Net Type Definition define the graphs that are legal Petri nets of this particular kind.

5.1.0.4 Many versions of Petri nets have many features in common and differ only in some features. In order to identify the same features in different Petri net types, PNML provides a mechanism for defining features separately, which can then be used in Petri Net Type Definitions. To this end, this International Standard defines a *Features Definition Interface*. This way, it is possible to define the legal labels representing a particular feature of a Petri net once and for all, and common aspects of a Petri net be exchanged among tools, even when the tools support different versions of Petri nets only.

5.1.0.5 Figure 1 gives an overview of the different parts of PNML and their relationships. The PNML Technology provides the Core Model, which

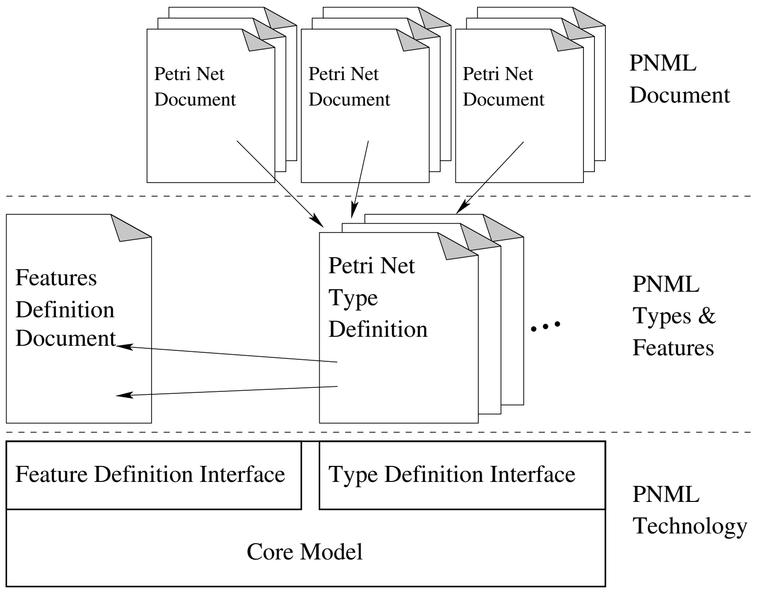

Figure 1: Overview on the structure of PNML

defines the general structure of a Petri net graph and all possible labels attached to it. The Features Definition Interface defines how to define particular labels. The Type Definition Interface defines how to define the labels belonging to a particular version of Petri net and which labels may or must be attached to which objects of the net. In a concrete Petri Net Type Definition, one can refer to the labels defined in a Features Definition Document so that different Petri net types may use the same features (with a common understanding). In a Petri Net Document, each Petri net must refer to a particular Petri Net Type Definition, which defines the features resp. labels it may contain.

5.1.0.6 Place/Transition nets and High-level Petri Net Graphs are defined in Clause 5.3 of this International Standard. New features and new types may be defined in the future.

5.2 Core Model

Editor's note: Initially, the UML diagram discussed in this clause was designed for providing an overview on the syntax of PNML only. Over the time it gained more an more meaning, as the discussions on the Core Model at the Brisbane meeting have shown. Right now, the UML model does not capture all requirements on PNML. For example, the condition that arcs can only connect nodes on the same page or the condition that a reference transition must refer to reference transition or to a transition is not formally captured in the UML models. It would be possible to express most of these additional conditions in OCL (the only condition that – to the best of my knowledge – cannot be expressed in OCL is that references must not be cyclic). Should we use OCL? Or is it sufficient to state the additional constraints in the text?

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

Editor's note: Do we need to refer to a concrete version of UML? In this and the following diagrams, it does not really matter.

5.2.1 Petri nets and objects.

5.2.1.1 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 will be an URL to the document with its Petri Net Type Definition (the structure of these documents will be defined in Clause 7.2).

5.2.1.2 A Petri net consists of *objects*, which, 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 attributes depend on the particular type of object (see Clause 5.2.3).

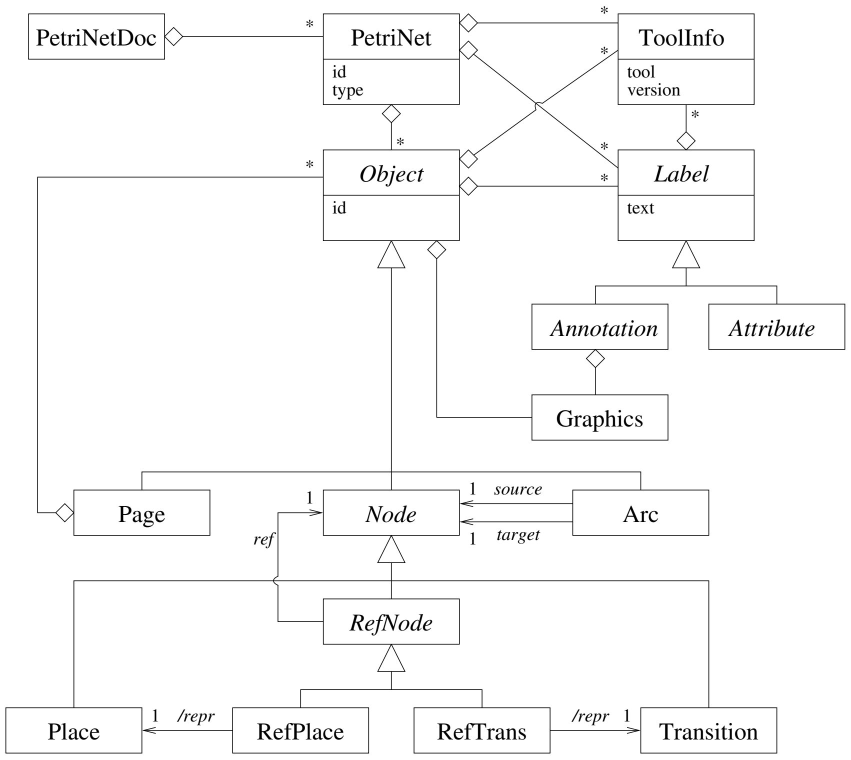

Figure 2: The PNML Core Model

5.2.1.3 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.

5.2.1.4 Note that it is legal to have an arc from a place to a place or from a transition to a transition according to this Core Model. The reason is that there are versions of Petri nets that support such arcs. So the Core Model must cover such arcs, since Petri Net Type Definitions can only restrict this Core Model. If a Petri net type does not support such arc, this restriction will be defined in the particular Petri Net Type Definition.

5.2.1.5 Three other kinds of objects are used for structuring a Petri net: *pages*; *reference places*; and *reference transitions*. A page may consist of other objects; they may even consists of further pages, which defines a hierarchy of subpages.

5.2.1.6 This International Standard requires that an arc must connect nodes on the same page only. The idea is that arcs connecting nodes on different pages do not have a graphical representation.

5.2.1.7 For establishing a connections between objects on different pages, a representative of this node may be drawn on another page. Then, this representative may be connected with nodes on this other page by arcs. Such a representative is called a reference node, because it has a reference to the node it represents (see Clause 5.2.5).

5.2.1.8 Since reference places and reference transitions enjoy basically the same structure, we generalize them to *reference nodes*.

5.2.2 Labels.

5.2.2.1 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 guard of a transition, or an arc annotation. In addition, the Petri net itself may have some labels. For example, the declarations of a HLPNG are included as a label of a high-level Petri net. The legal labels and the legal combinations of labels are defined by the *Petri Net Type*. The type of a Petri net is determined by a unique reference to a *Petri Net Type Definition* (PNTD), which is discussed in Clause 7.2.

5.2.2.2 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 inscriptions, transition guards, and timing or stochastic information.

Editor's note: 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. But, I am not sure whether we should say this here, because the mapping of concepts to XML syntax is only later.

5.2.2.3 In contrast, an attribute is not displayed as a 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 define the effect on the graphical appearance of an attribute.

Editor's note: This Standard could define some Standard Features including some guidelines on there graphical representation.

The Features Definition Interface could provide a mechanism for defining the graphical appearance of an attribute. But, this would mean some extra work. So, the Brisbane comments suggest that we should not include this mechanism here

5.2.2.4 Note that Label, Annotation and Attribute are abstract classes in the Core Model, which means that they do not define concrete labels, annotations, and attributes here. The concrete labels are not defined in the Core Model. The concrete labels will be defined in the Features Definition and will be used in Petri Net Type Definitions (see Clauses 7.1 and 7.2).

Editor's note: At the Brisbane meeting there was some discussion on these abstract classes. The above text should clarify this point. Concrete classes come in when defining a concrete Petri Net Type or a Feature.

Another way to express this in UML would be interfaces, which must be implemented by the Petri Net Type Definition.

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

5.2.3 Graphical information

5.2.3.1 Graphical information is 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 and 6.2 for details).

Editor's note: 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. This could be included somewhere in this standard. But, I feel that it is much to detailed to be included right here.

5.2.4 Tool specific information (ToolInfo)

5.2.4.1 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 format 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.

5.2.4.2 The same object may be equipped 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 referring to other objects of the net, because the deletion of other objects by a different tool might make this reference invalid and leave the tool specific information inconsistent. Clearly, this intended use of the tool specific information cannot be enforced; but it can serve as a guideline for its purpose and its use.

Editor's note: At the Brisbane meeting, there was a discussion that the tool specific information should be precisely defined within the meta model. But, the idea of the tool specific information is that we do not know what a tool might want to store here. So we cannot fix a concrete structure. So, the tool may use any structure it likes (within the tool specific information). Maybe, we should make this even more explicit by making this class abstract. Then, it would be clear that the tool must provide its own implementation. Another possibility would be making it an interface.

5.2.5 Pages and reference nodes.

5.2.5.1 For structuring large Petri nets, PNML supports a page concept: Different parts of a net may be split into separate *pages*. A page is an object that may consist of other pages or objects. An arc, however, may connect nodes on the same page only[1]. A *reference node*, which can be either a *reference transition* or a *reference place* represents an appearance of a node. It can refer to any node on any page of the Petri net as long as there are no cyclic references among reference nodes; this guarantees that, ultimately, each reference node refers to exactly one place or transition of the Petri net, which is indicated by the derived association */ref*.

[1]The reason is that an arc cannot be drawn from one sheet of paper to another when printing the different pages.

5.2.5.2 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.

Editor's note: There was a comment that we should not include pages in Part 2 of this International Standard. I left this concept in because, this concept does not include any difficulties. Moreover, the Petri net experts at Bologna strongly recommended to have it in the Standard (there was even strong support for modules). As argued earlier having an transfer syntax not even supporting pages would be of no use at all.

Since pages and references are a stable concept in PNML, there is no reason for assuming that they could result in additional delays in finalising this Standard.

5.3 Petri Net Type Models

5.3.0.3 This Clause defines meta models for three versions of Petri nets: Place/Transition Systems, Well-Formed Nets (WFNs), and High-Level Petri Net Graphs (HLPNGs). These meta models are called Type Models in order to distinguish them from the PNML Core Model.

5.3.0.4 Well-Formed Nets and High-Level Petri Net Graphs have the same underlying model. The only difference is in the built-in sorts and operations and in the built-in mechanism for defining new Sorts. So we present a general model for High-Level Nets first and then discuss the built-in Sorts and Operations for Well-Formed Nets

Editor's note: Conceptually, Place/Transition-Systems can be considered as a special version of HLPNGs (as defined in Part 1 of this International Standard). But, there are syntactical differences! The transfer format for Place/Transition-Systems should not force tools not supporting HLPNGs to use the syntax of HLP-NGs, which is much more complicated than the one typically used for simple Place/Transition-Systems. Therefore, Place/Transition-Systems are not a special case of HLPNGs on the syntactical level!

This Standard could, however, define a transformation between these two formats.

5.3.1 Place/Transition Systems

5.3.1.1 We start with the Type Model for Place/Transition systems. A Place/Transition-System is a Petri Net Graph, where the places and transitions have names, and where each place is labelled with a natural number (the Initial Marking), and each arc is labelled with a non-zero natural number (the Arc Annotation). The details are defined in Part 1 of this International Standard.

5.3.1.2 Figure 3 shows the Type Model for Place/Transition-Systems.

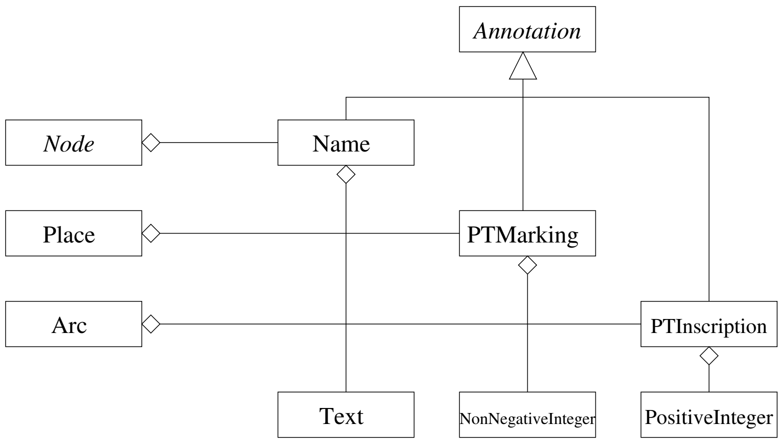

Figure 3: The Type Model of Place/Transition Systems

5.3.1.3 This Type Model basically, defines that each node of the Core Model must have an Annotation *Name* that consists of a *Text* (in addition to the standard information for annotations such as graphical and tool specific information, which are defined in the Core Model). In addition, each Place must have an Annotation *PTMarking* that consists of a *NonNegativeInteger*, and each Arc must have an Annotation *PTInsciption* that consists of a *PositiveInteger*.

5.3.1.4 Remember that the UML Meta Models do not yet fix the XML syntax for these elements. The classes Text, NonNegativeInteger, and PositiveInteger will be immediately mapped to RELAX NG constructs or to the corresponding XML Schema data types (see Clause 7 for details).

5.3.1.5 Actually, the only classes defined here are Name, PTMarking, and PTInscription. The classes Node, Place, Arc, and Annotation come from the 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-Systems.

Editor's note: This Type Model does not yet exclude arcs from places to places or from transitions to transitions. This could be expressed in the text or it could be formalized in OCL. What should we do?

5.3.2 High-Level Petri Net Graphs

5.3.2.1 In this Clause the Type Model for High-Level Petri Net Graphs will be defined.

5.3.2.2 A HLPNG is a Petri Net Graph that is equipped with a Declaration that defines Sorts, Functions, and Variables that are the basis for defining Terms. The Declaration itself will become an Annotation of the Petri Net Graph itself; the Sorts will used as an Annotation defining the Domain of the Place; the Terms will be used for representing the Initial Marking of each Place, the Arc Annotations, as well as the Transition Guards. The details are defined in Part 1 of this International Standard.

5.3.2.3 Figure 4 shows the first part of the Type Model, which defines the structure of a Signature. A Signature consists of Sort Declarations, Variable Declarations, and Function Declarations. In addition to the Sorts that are defined within the Signature (UserDefined), there can be other Sorts: BuiltIn Sorts, the MultisetSort corresponding to another sort, and ProductSorts of a sequence of existing Sorts. A list of BuiltInSorts for HLPNGs

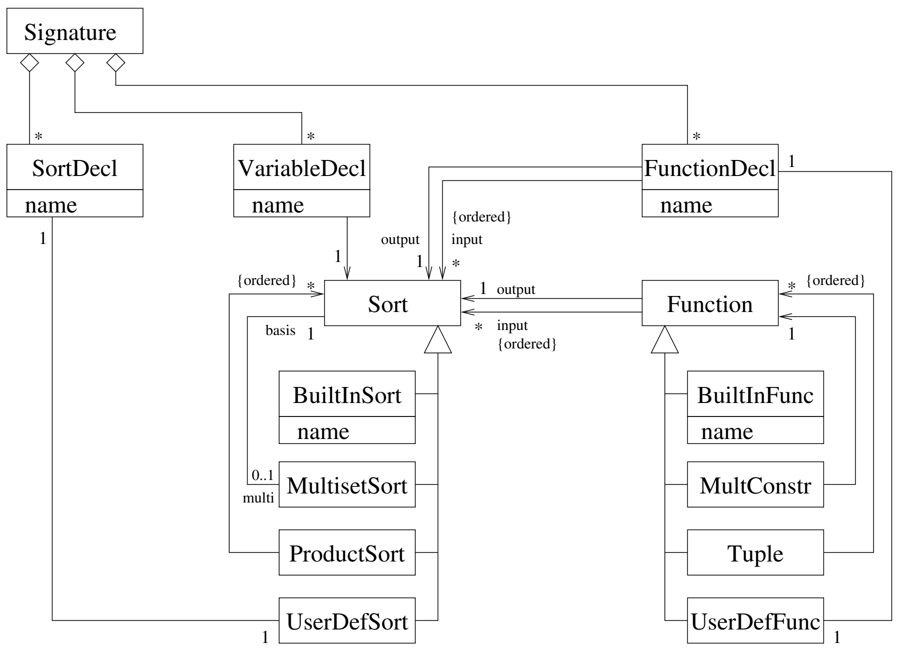

Figure 4: The model for Signatures

will be given later. Likewise, there can be Functions that are defined in the Signature (UserDefinedFunctions) and there are BuiltInFunctions: Functions that convert a sequence of values of a basis set to the corresponding Multiset, which is called a MultisetConstructor. Moreover, there are Function for building Tuples (a product) from elements of the component Sorts. Each Function has a sequence of input Sorts as well as a sequence of output Sorts. For the UserDefined Functions the input Sorts and the Output Sorts are defined in the FunctionDeclaration. For the MulitsetConstructors, there is a restriction: All the elements of the sequence of input Sorts refer to the same Sort and the output Sort must be the multiset Sort corresponding to this Sort. For the Type Function, the output Sort must be the ProductSort of the sequence of input Sorts.

Editor's note: This requirements could be expressed in OCL. Should we do that?

5.3.2.4 For a particular Signature, we can construct Terms as defined in Part 1 of this International Standard. Terms are captured in the UML diagram of Fig. 5. A Term can be built up from Variables and Function-

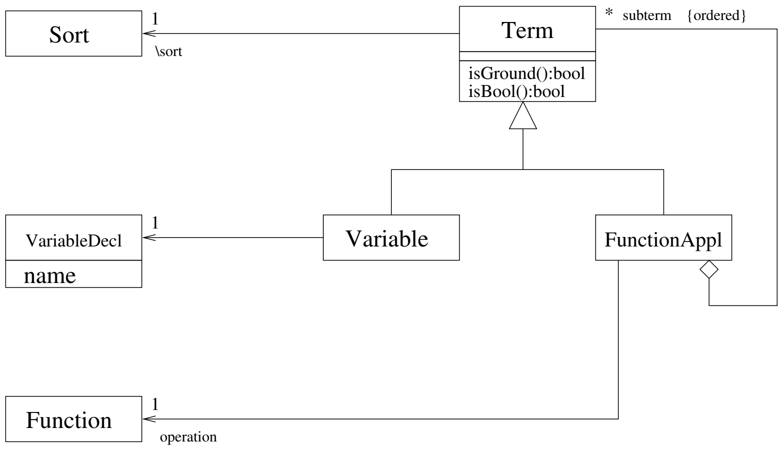

Figure 5: The model for Terms

Applications on subterms. The sort of a Term can be derived from the Sort of the corresponding Variable or the output Sort of the Function. For this model, there is an additional requirement that the sequence of sorts of subterms in a FunctionApplication must be the sequence of input Sorts of the corresponding Function.

Editor's note: Again, this requirements could be expressed in OCL.

5.3.2.5 In Part 1 of this International Standard, special Terms have been defined. These are Ground Terms, which are Terms without Variables. And there are Boolean Terms, which are Terms of builtin sort Boolean. This is reflected in the methods isGround and isBool, which return true if the Term is a Ground Term or a Boolean Term respectively.

5.3.2.6 Based on these Models for Signatures and Terms, Fig. 6 defines the Type Model for High-Level Petri Net Graphs. Again Nodes, have an An-

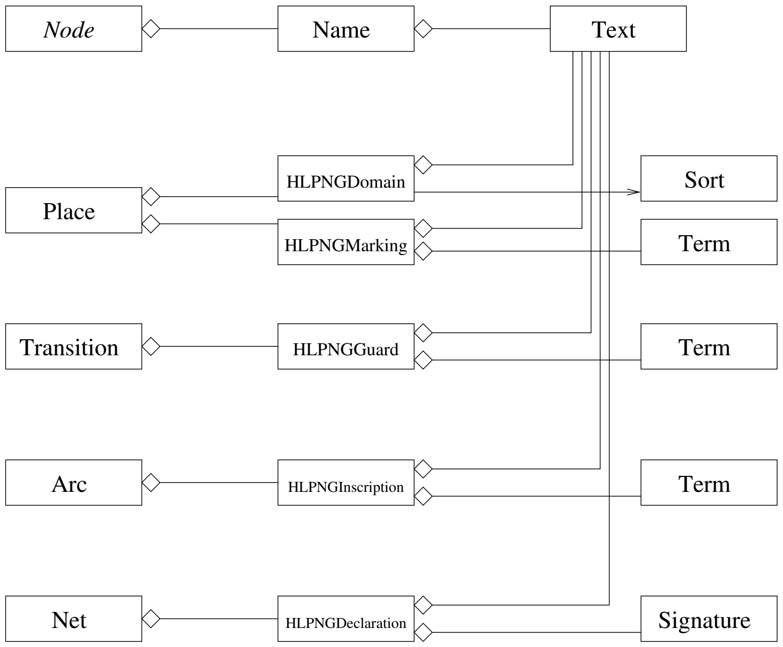

Figure 6: The Type Model of HLPNGs

notation with their name, the net itself has a Declaration as its Annotation, which is a Signature (as defined in Fig. 4). Note that this model defines an abstract syntax for Signatures only. In order to allow tools to store some concrete text, the Annotation has also a text, which should be the same Signature in concrete syntax. The concrete syntax, however, will not be defined in this International Standard.

The same applies to all other labels defined below.

5.3.2.7 A Place has two Annotations: A HLPNGDomain, which defines the Sort of the place, and a Term, which defines the initial Marking. This Term is required to be of the multiset Sort of the Sort of the Place; moreover, it must be a Ground Term.

Editor's note: Again, this requirements could be expressed in OCL.

5.3.2.8 A Transition has one Annotation: a HLPNGGuard, which is the Transition Guard, i.e. a Term of sort Boolean.

5.3.2.9 An Arc also has one Annotation: a HLPNGInscription, which is a Term, which has the multiset Sort of the Sort of the Sort associated with the corresponding Place of the arc.

5.3.2.10 In addition to the Annotations defined above, the Type Model for HLPNGs requires that arcs must not connect two places and must not connect two transitions.

5.3.3 BuiltIn Sorts and Functions

5.3.3.1 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.

Editor's note: 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]).

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

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

5.3.3.3 HLPNGs

- 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: 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: Products are included explicitly in the Meta Model; so it is not necessary to include this here

  Editor's note: 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.

5.3.3.4 WFN

Editor's note: This should be contributed by AFNOR based on their proposal of WFNs.

Basically WFN have the singleton sort and all kinds of finite subsets of integers, with predecessor, successor operations and comparison operations. Moreover, there are the Booleans. For each Sort, there is a Function (Constant) that denotes the multiset in which each element occurs exactly once (broadcast).

6 PNML Syntax

6.0.3.1 In Clause 5, we have defined the PNML Core Model and we have shown how Type Models can be obtained extending the Core Model. These Meta Models, however do not immediately provide an XML syntax for exchanging such models.

6.0.3.2 In this Clause, the XML syntax of PNML will be defined, i. e. it will be defined how the Meta Model concepts of PNML are mapped to XML. Here, we give an immediate translation from the concepts of PNML to XML. Annex A, gives a formal definition of valid PNML documents in terms of RELAX NG technology.

Editor's note: Today, there are different Techniques for mapping Meta Models immediately to XML. MOF and XMI are probably the most prominent candidates. There has been a discussion with AFNOR experts on using XMI for mapping the PNML Meta Model(s) to XML. This sound appealing on a first glance. But, there are two problems with this approach:

1. The resulting XML syntax will be quite different from the syntax of PNML as it was up to now. Therefore, tools implementing the current version of PNML will have to completely reimplement their export and import modules. Moreover existing PNML documents will not be readable with the new PNML. This will strongly decrease the acceptance of this Standard.

2. I (E. Kindler) am no an expert in XMI. But, I have talked to several experts concerning the following problem: One major objective of PNML was to support the exchange of Petri nets among tools even if the tools do not support exactly the same version of Petri Nets (i.e. with slightly variants in the underlying Meta Model). The experts I have been talking to told me that this is not possible with XMI.

Given these problems, using XMI might be an option some time in the future, but it is not an option right now.

Right now, we use the RELAX NG technology for defining the syntax of PNML. The reason was that, at the time when PNML was developed, RELAX NG appeared to be the most mature technology. Today XML Schema could be used also (actually, we are using XML Schemas data types). But, it would require much extra effort, to make everything work (and I don't know whether we have an expert who could and would do that). If we want to publish this Standard soon, RELAX NG seems to be the only option.

As long as only concrete Petri Net Types are considered, it does not make any difference whether the PNML syntax is defined in terms of RELAX NG or in terms of XML Schema or by a simple DTD (except for the fact that DTD cannot capture some constraints that can be captured by XML Schema or RELAX NG). But, it makes a difference when own PNTDs should be defined, because the PNTDs must be formulated as RELAX NG grammars (or as XML Schemas). This ist where RELAX NG or XML Schema come in explicitly.

6.0.3.3 This Clause is only concerned with the mapping of the concepts of PNML Core Model concepts to PNML. The mapping of Type Models will be defined along with the Petri Net Type Definition through RELAX NG grammars, which will be discussed in Clause 7.

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

6.1 PNML Core Model

The PNML Core Model is translated to XML syntax in a straightforward manner.

6.1.1 PNML elements.

6.1.1.1 Basically, each concrete class[2] of the PNML Core Model is translated to an XML element. This translation 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 aggregations of the Core Model define in which elements it may occur as a child element.

6.1.1.2 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. For instance, the attribute `ref` of a reference place must transitively refer to a place of the same net. The set to which a reference is restricted, is indicated in the table (e.g.).

Editor's note: The RELAX NG grammar for PNML cannot capture this requirement right now. But, in order to conform to this standard, these restrictions must be met, too.

At the Petri Net Conference in Eindhoven (June 2003), there was a wide agreement that it is acceptable to have a syntax definition

[2]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>`	

with some technology (RELAX NG or XML Schema) even if some requirements cannot be formally captured. Actually, even when using XMI and OCL there remain some conditions that cannot be formally captured. These requirements can and must be stated in this or other Standard documents.

6.1.2 Labels

6.1.2.1 There are no PNML elements for labels because the Core Model does not define any concrete labels. Concrete labels are defined by the Petri Net Types. An XML element that is not defined by the 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 inscription. A legal element for a label may consist of further elements. The value of a label appears as a string in a `<text>` element. Furthermore, the value may be represented as an

Table 2: Elements in the <graphics> element depending on the parent 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>

XML tree in a <structure> element[3]. An optional PNML <graphics> element defines its graphical appearance, and further optional PNML <toolspecific> elements may add tool specific information to the label.

6.1.3 Graphics

6.1.3.1 PNML elements and labels include graphical information. The structure of the PNML <graphics> element depends on the element in which it occurs. Table 2 shows the elements which may occur in the substructure of a <graphics> element.

6.1.3.2 The <position> element defines an absolute position and is required for each node, whereas the <offset> element defines a relative position and is required for each annotation. The other sub-elements of <graphics> are optional. For an arc, the (possibly empty) sequence of <position> elements defines its intermediate points. 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. More details on the effect of the graphical features can be found in Sect. 6.2.1.

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

6.1.4 Example

Editor's note: 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.

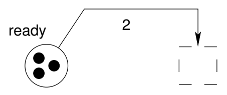

Figure 7: A simple Place/Transition-System

6.1.4.1 In order to illustrate the structure of a PNML document, we given an example PNML document representing the Petri net shown in Fig. 7. Listing 1 shows the corresponding PNML Document. It is a straightforward translation, where we have labels for the names of objects, for the initial markings, and for arc inscriptions. Note that we assume that the dashed outline of the transition results from the tool specific information `<hidden>` from an imaginary tool *PN4all*. This tool specific information says, that in this tool *PN4all*, this particular transition should be hidden. Since hiding a node is no concept of PNML, the tool is forced to encode this information into a tool specific element. The tool has chosen to use an element `<hidden>`.

Editor's note: I am looking forward to better examples for tool specific information.

Listing 1: PNML code of the example net in Fig. 7

```
<pnml xmlns="http://www.example.org/pnml">
  <net id="n1" type="http://www.example.org/pnml/PTNet">
    <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>
      </initialMarking>
    </place>
    <transition id="t1">
      <graphics>
        <position x="60" y="20"/>
      </graphics>
      <toolspecific tool="PN4all" version="0.1">
        <hidden/>
      </toolspecific>
    </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>
  </net>
</pnml>
```

6.2 Graphical representation (non-normative)

6.2.0.1 In this Clause, we discuss the graphical features of PNML and their effect on the graphical presentation of a PNML document. Clause 6.2.1 gives an informal overview of all graphical features. Clause 6.2.2,defines the graphical presentation of a PNML document by an XSLT transformation from PNML to the *Scalable Vector Graphics* (SVG).

Editor's note: If the graphical appearance of a PNML Net is considered to be non-normative, it might be sufficient to include the informal overview. Clause 6.2.2 could be deleted. If we decide to have it in the Standard, it can be made more formal (with some support from the resp. expert) and the Appendix could include the XSLT stylesheet.

6.2.1 Graphical appearance

6.2.1.1 Table 3 lists the graphical elements that may occur in the PNML `<graphics>` element along with their attributes. 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.

6.2.1.2 The `<position>` element defines the absolute position of a net node or a net annotation, where the x-coordinate runs from left to right and the y-coordinate from top to bottom. The `<offset>` element defines the position of an annotation relative to the position of the object.

6.2.1.3 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 points act as line connectors or Bezier control points.

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

6.2.1.4 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.

6.2.1.5 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.

6.2.1.6 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.

6.2.2 Transformation to SVG

6.2.2.1 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 visualise Petri nets using other means than SVG, but the SVG translation can act as a reference model for such visualisations. 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.

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

6.2.2.2 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.

6.2.2.3 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.

6.2.2.4 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.

6.2.2.5 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.

6.2.2.6 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.

6.2.2.7 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.

6.2.2.8 Type specific transformations. Above, we have discussed a transformation that works for all PNML documents, where all annotations are displayed as text. For some Petri net types, one would like to have other graphical representations for some annotations. This can be achieved with customized transformations. The technical details of customized transformations are not yet fixed. Due to the rule-based transformations of XSLT, equipping the Type Definition Interface and the Feature Definition Interface of PNML with some information on their graphical appearance seems to be feasible. Basically, each new feature can be assigned its own transformation to SVG. Adding these transformations to the standard ones for PNML gives us a customized transformation for this Petri net type.

Editor's note: At some point, we need to talk about how to get rid of pages and reference nodes, because these are concepts that are not present in HLPNGs and Place/Transition nets. The idea is simple: 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]. Where does this belong to?

7 Features and Type Definition Interfaces

7.0.2.1 This Clause defines how to extend the Labels of the PNML Core Model and how new Petri Net Types can be defined by using the RELAX NG technology. In these definitions, the RELAX NG technology plays two roles at a time: First we use it as syntactic representation of the Meta Model (to be more precise the Type Model); i.e. we define the structure of the Label. Second, we use it for defining the mapping from the Type Model to XML syntax.

7.1 Feature Definition Interface

7.1.0.1 In this Clause, we show how new labels can be defined in PNML, by the help of the Features Definition Interface.

To this end, we consider the initial marking of places of Place/Transition-Systems again. Figure 8 shows a section of the Type Model for Place/Transition-Systems again, where PTMarking is defined as an Annotation for places.

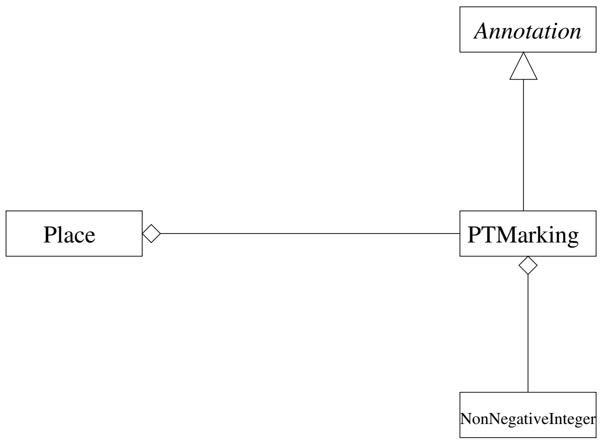

Figure 8: Example: Annotation PTMarking

Listing 2: Label definition

```
<define name="PTMarking"
  xmlns:pnml="http://www.informatik.hu-berlin.de/top/pnml">
  <element name="initialMarking">
    <interleave>
      <element name="text">
        <data type="nonNegativeInteger"
          datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
      </element>
      <ref name="pnml:StandardAnnotationContent"/>
    </interleave>
  </element>
</define>
```

7.1.0.2 Listing 2 shows the RELAX NG definition of this PTMarking. Line one starts with the definition of PTMarking. Line two refers to the PNML namespace. Line 3 states that the XML syntax for a PTMarking is a `<initialMarking>` element[4]. Lines 4 to 6 define the contents of this element. There is an element `<text>` which contains a non-negative integer, a data type defined in XML Schema. Line 9 includes all the standard information of Annotations. Note that RELAX NG does not support explicit inheritance; the `<ref>` statement is the RELAX NG counterpart to inheritance. It guarantees that the PTMarking has all graphical and tool specific information, defined for Annotations in the Core Model.

7.1.0.3 The Annotation defined above is used in lines 16–18 of List. 1.

7.1.0.4 The translation of other Concrete Labels works in the very same way. Remember that, i addition to the information given in the Type Model, the RELAX NG grammar must provide the XML syntax for the elements representing it.

7.1.0.5 Such label definitions can either be given explicitly for each Petri net type, or they can be included in the Conventions Document, such that Petri net type definitions can refer to these definitions. The example from Listing 2 was taken from the Conventions Document (see Appendix B).

7.2 Type Definition Interface

7.2.0.1 This Clause shows how to define a Petri Net Type by RELAX NG technology. As an example, we choose Place/Transition systems as defined by the Meta Model shown in Fig. 3.

7.2.0.2 Listing 3 shows the Petri Net Type Definition (PNTD) as a RELAX NG grammar. Firstly, it includes both the definitions of the PNML Core Model (`pnml.rng`) and the definitions of the Conventions Document (`conv.rng`) (see Appendix B), which, in particular, contains the definition

[4]Note that there is no standard name derived from the PTMarking; the concrete XML syntax for this concept is encoded explicitly in the RELAX NG grammar.

Listing 3: PNTD for P/T-Systems

```
<grammar ns="http://www.example.org/pnml"
         xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:conv="http://www.informatik.hu-berlin.de/top/pnml/conv">
  <include href="http://www.informatik.hu-berlin.de/top/pnml/pnml.rng"/>
  <include href="http://www.informatik.hu-berlin.de/top/pnml/conv.rng"/>
  <define name="NetType" combine="replace">
    <text>http://www.example.org/pnml/PTNet</text>
  </define>
  <define name="Net" combine="interleave">
    <optional><ref name="conv:Name"/></optional>
  </define>
  <define name="Place" combine="interleave">
    <interleave>
      <optional><ref name="conv:PTMarking"/></optional>
      <optional><ref name="conv:Name"/></optional>
    </interleave>
  </define>
  <define name="Arc" combine="interleave">
    <optional><ref name="conv:PTArcInscription"/></optional>
  </define>
</grammar>
```

from List. 2, a similar definition for arc inscriptions of P/T-systems, and a definition for names.

7.2.0.3 Secondly, the PNTD defines the legal labels for the whole net and the different objects of the net. In our example, the net and the places may have an annotation for names. Furthermore, the places are equipped with an initial marking (PTMarking) and the arcs are equipped with an arc inscription (PTArcInscritption). Note that all labels are optional in this type. The labels are associated with the net objects by giving a reference to the corresponding definitions in the Conventions Document. Technically, the definition extends the original definition of the net, places and arcs of the RELAX NG grammar for PNML.

Editor's note: This needs a more careful explanation and a more precise definition how a RELAX NG grammar for a PNTD must look like.

I hope that M. Weber can provide some support here.

A RELAX NG Grammar for PNML (Normative)

This appendix gives a complete definition of the PNML Core Model 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 some syntactical restrictions of the Core Model cannot be expressed in RELAX NG. Still, these restrictions are mandatory for valid PNML documents.

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: 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.

B RELAX NG Grammar for Conventions (Normative)

This Appendix is a first draft of the Conventions Document with some Standard Feature Definitions, that will be used in the PNTDs for Place/Transition-Systems and for High-Level Petri Net Graphs.

```
<?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>
```

C RELAX NG Grammars for PNTDs (Normative)

C.1 Place/Transition-Systems

```
<?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: There seems to be a slight difference between the Type Model of Place/Transtion-Systems 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.

C.2 HLPNGs

Editor's note: This is not yet the RELAX NG grammar for HLPNGs! This is just an outline, how the definition could look like. But the part making it a PNTD is still missing.

The work on the RELAX NG grammar should be started (with some support from F. Kordon, M. Westergaard, and M. Weber), when the concepts of the Type Model for HLPNGs are fixed (cf. Clause **??**) and RELAX NG is the the technology of choice.

```
<!-- A declaration will go as an annotation of the net.
     It defines the sorts, operations and variables used in
     the net. Note that a sort may be made a subsort of a
     supersort by using attribute super.

     Furthermore, a declaration may include all kinds of
     definitions, which are considered as a comment by now.
```

```
    We will assume that some sorts (boolean and naturals)
    and some operations ( +, <=, on naturals and multisets
    are predefined).  Moreover, we assume that with the
    defined sorts we have also the tuples over these sorts
    and multisets over each sort.
-->

 <define name="DECL">
   <element name="declaration">
     <zeroOrMore>
       <choice>
         <ref name="SORTDECL"/>
         <ref name="OPDECL"/>
         <ref name="VARDECL"/>     LP: why do VARDECL and
         <ref name="SORTDECL"/>        SORTDECL appear twice?
       </choice>
     </zeroOrMore>
   </element>
 </define>

 <define name="SORTDECL">
   <element name="sortdecl">
     <attribute name="name">
       <data type="xsd:ID"/>
     </attribute>
     <optional>
       <attribute name="super">
         <!-- a possible supersort -->
         <data type="xsd:IDREF"/>
       </attribute>
     </optional>
   </element>
 </define>

 <define name="OPDECL">
   <element name="opdecl">
     <attribute name="name">
       <data type="xsd:ID"/>
```

```
      </attribute>
      <optional>
        <element name="parameters">
          <oneOrMore>
            <ref name="SORT"/>
          </oneOrMore>
        </element >
      </optional>
      <ref name ="SORT"/>
    </element>
  </define>

  <define name="VARDECL">
    <element name="vardecl">
      <attribute name="name">
        <data type="xsd:ID"/>
      </attribute>
      <ref name ="SORT"/>
    </element>
  </define>

  <define name="DEFINITIONS">
    <element name="definitions">
      <anyString/>
    </element>
  </define>

<!-- A sort will go as an annotation of places. The marking of
     the place will be a multiset over this sort.  Note that the
     sort may be a multiset sort; in that case the markings is
     a multiset over multisets.

     Currently, I don't know how to check this requirement in TREX.
     In XML-Schema this could be checked by a keyref constraint.
 -->

  <define name="SORT">
    <choice>
      <!-- We could admit the definition of sorts, by simply
           using them.
         <ref name="SORTDECL"/>
        -->
      <ref name="MSSORT">
      <element name="sort">
        <attribute name="name">
          <data type="xsd:IDREF"/>
        </attribute>
      </element>
    </choice>
  </define>

  <define name="MSSORT">
      <element name="mssort">
        <ref name="SORT/>
      </element>
  </define>

 <!-- A multiset term will go as an annotation of
      arcs (arc inscription). Its sort must be a
      multiset over
   -->

  <define name="MSTERM">
    <!-- I don't know how to impose the requirement on MSTERMS
         that they have a multiset sort in TREX;  I hope Michael
         will help me with that.
      -->
    <ref name="TERM"/>
  </define>

  <define name="TERM">
    <choice>
      <ref name="VAR"/>
      <ref name="OPAPP"/>
    </choice>
  </define>
```

```
  <define name="VAR">
    <element name="var">
      <attribute name="name">
        <data type="xsd:IDREF"/>
      </attribute>
    </element>
  </define>

  <define name="OPAPP">
   <element name="op">
      <attribute name="name">
        <data type="xsd:IDREF"/>
      </attribute>
      <optional>
        <!-- currently the sorts of the arguments are not
             checked with respect to the parameter sorts;
             I don't know how to do this in TREX.  In XML
             Schema, this could be done by a keyref constraint
          -->
        <element name="arguments">
          <zeroOrMore>
            <ref name="TERM"/>
          </oneOrMore>
        </element >
      </optional>
  </define>

<!-- A boolean term will go as an annotation of
     transitions (transition guard).
   -->

  <define name="BOOLTERM">
    <!-- I don't know how to impose the requirement on a BOOLTERM
         that it must have (built-in) sort boolean in TREX;
         I hope Michael will help me with that.
      -->
    <ref name="TERM"/>
```

```
</define>
```

D Guidelines for Introducing New Net Classes

Editor's note: Yet to come.

References

[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. 17

[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, 24th International Conference*, volume 2679 of *LNCS*, pages 483–505. Springer, June 2003. 0.0.0.1

[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, Final Committee Draft, May 2002.

[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. 6.2.2.8

[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. 0.0.0.1, 5.3.3.1

[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. 0.0.0.1

[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: These references are very prelimary. The strong bias on my (E.K.) own publications should be eliminated.