High-level Petri Nets – Transfer Syntax

– Proposal for the International Standard ISO/IEC 15909-2 –

Draft Version 0.3.0

Ekkart Kindler

April 21, 2004

Contents

Preface

This is a draft version of a proposal for an International Standard (ISO/IEC 15909-2) for a Transfer Syntax for High-level Petri Nets. It is based on [22, 4] and supposed to stir the discussion on this upcoming standard – formally and informally.

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

Ekkart Kindler, Paderborn, April 2004

1 Scope

This International Standard defines a transfer syntax (interchange format) for Petri nets, which is based on XML: The *Petri Net Markup Language* (*PNML*). PNML facilitates the exchange of Petri nets among different Petri net tools.
There are many different versions and variants of Petri nets. High-level Petri Nets as defined in ISO/IEC 15909-1 are but one version. The transfer syntax defined in this International Standard is independent from the particular version or variant of Petri nets and supports the exchange of any version of Petri nets. This International Standard defines techniques for the definition of different extensions (features) of Petri nets and for the definition of different versions of Petri net types: *Petri Net Type Definitions* (*PNTDs*).
This International Standard gives a Petri Net Type Definition for Place/-Transition Nets and High-level Petri Nets as defined in ISO/IEC 15909-1. This way, it defines a transfer syntax for High-level Petri Nets. The definition of extensions of Petri nets and of other versions of Petri nets will be defined in ISO/IEC 15909-4.
This International Standard also defines 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 defines in ISO/IEC 15909-3.

`< maybe, we should include modular PNML already into this Standard`

>

2 Normative References

ISO/IEC 15909-1
XML
XML Schema
XSLT
CSS
SVG
RELAX NG
UML
SVG

3 Terms and Definitions

3.1 General

For the purpose of this International Standard, the following terms and definitions are used.

3.2 Glossary

Annotation A label that is graphically represented as a text near the corresponding object.

Arc A directed edge that connects two nodes of a graph. Typically, an arc is graphically represented by an arrow. The start node of the edge is called the source of the edge, the end node is called the target node of the edge. In most types of Petri nets, an arc may connect a place to a transition or a transition to a place only.

Attribute A label that is graphically represented in the form or shape of the corresponding object.

Feature Definition A feature is a particular extension of a Petri net. The same feature may occur in different versions of Petri nets. For a uniform use, this International Standard provides a technique for defining features: the Features Definition Interface.
A feature definition that is defined in some (informal or formal) standard is called a Standard Feature Definition.

Graphical Information Each object of a Petri net as well as annotations of a Petri net may be equipped with information on its graphical representation. The type of graphical information depends on the object. But, each object has an absolute position (on the page it belongs to) and each annotation has a relative position (with respect to the object it correspond to).

Global Label A label that is not attached to an object but is attached to the Petri net itself.

Label Each object of the Petri net may carry additional information. This information is attached to the corresponding object as a label. Moreover the Petri net itself may have labels which are called global labels.
Typically, labels are graphically represented by a text near to the corresponding object. In that case, the label is called an annotation.
Sometimes the annotation is not represented as a text, but it is represented in the shape of the corresponding arc. In that case, the label is called an attribute.

Node The nodes of a Petri net are its places and its transitions. In Petri nets with pages, there are also reference places and reference transitions, which are also considered as nodes.

Object (of a Petri net) The arcs, places and transitions of a Petri net. In a Petri net with pages, also the pages and the reference places and reference transitions are objects of this net.

Page For representing large Petri nets, the net may be split into different pages. Each node of the Petri net belongs to a page, and arcs may only connect nodes that belong to the same page. In order to connect nodes on different pages, reference nodes can be used.

Petri Net A Petri net consists of two types of nodes, the places and the transitions and arcs that connect these nodes. Additional information can be attached to all objects of the Petri net by labels. For structuring purposes, a Petri net may also consist of pages and reference nodes.

Petri Net Document A document that contains one or more Petri nets.

Petri Net Type Definition (PNTD) There are many different versions and variants of a Petri net. The Petri Net Type Definition Interface is a technique for defining the syntax of a particular version of Petri net. A Petri Net Type Definition, can refer to features defined in the Feature Definitions from some Feature Definition Document.
A Petri Net Type Definition that is published in this or other (formal or informal) standards is called a Standard Petri Net Type Definition.

Reference Node A reference place or a reference transition. A reference node is used in Petri nets with pages as a representative of a node defined on another page.

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

Reference Transitions A reference node that is a place. Its reference must point to another reference transition or to a transition.

Source Node The start node of an arc.

Target Node The end node of an arc.

Tool Specific Information Each object of a Petri net can be equipped with information that is specific to a particular tool and is not meant to be used by other . This information is called tool specific information.

3.3 Abbreviations

`< here is a list of important abbreviations; the definitions will be added later >`
- PNML
- PNTD
- RELAX NG
- SVG
- UML
- XML
- HLPNGs

4 General Principles

4.1 Overview

This International Standard defines a transfer syntax for different versions and variants of Petri nets. This transfer syntax should be not restricted to particular versions and variants of Petri nets. In particular, it should be open for future extensions and future versions of Petri nets. In order to obtain this flexibility, the transfer syntax, basically, represents a Petri net as a labelled graph, where all type specific information of a net is attached as a label to some node or to some arc of the net. This basic structure is called the PNML *Meta Model*, which will be discussed in Clause 4.2.
The PNML Meta Model has no restrictions concerning the labels attached to the net and its objects. Therefore, it is apt for representing any kind of Petri net. In order to give a unique meaning to the attached labels, the used labels must be defined somewhere. To this end, PNML provides two mechanism: the *Features Definition Interface* for defining legal labels and the *Type Definition Interface* for defining new Petri net types.
Some features of Petri nets and some types will be defined in Clause 8 of this International Standard. However, there can be new features and new types defined in the future.
Figure 1 gives an overview on the different parts of PNML and their relation.

`<more information to be added>`

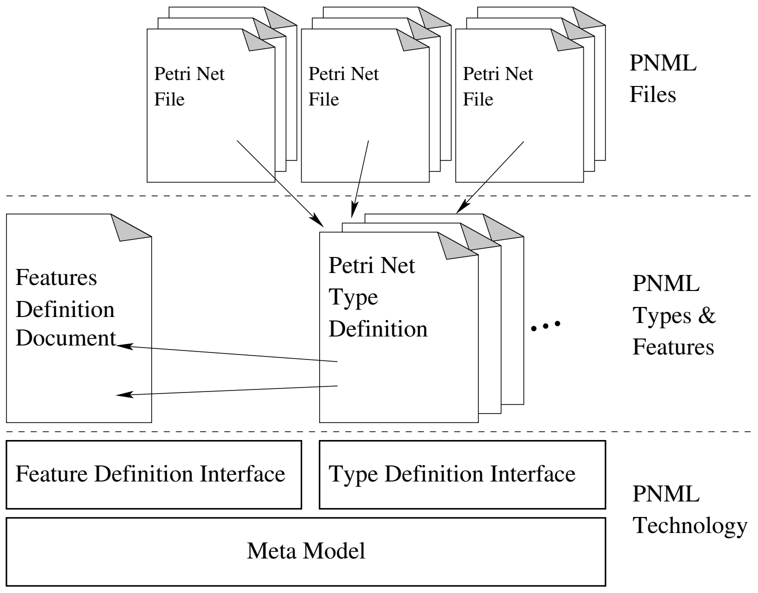
Petri Net File
Petri Net File
Petri Net File
PNML Files
Features Definition Document
Petri Net Type Definition
...
PNML Types & Features
Feature Definition Interface
Type Definition Interface
Meta Model
PNML Technology

Figure 1: Overview on the structure of PNML

4.2 Meta Model

Figure 2 shows the PNML Meta Model as an UML class diagram. This meta model will be discussed in the following Clauses.

4.2.1 Petri nets and objects.

A document that meets the requirements of PNML is called a *Petri net document*; it may contain several *Petri nets.* Each Petri net consists of *objects*, which, basically, represent the graph structure of the Petri net[1]. Each object within a Petri net document has a unique *identifier*, which can be used to refer to this object. An object is a *place*, a *transition* or an *arc.* For convenience, a place or a transition is called a *node.*
For structuring a Petri net, there are three other kinds of objects, which will be explained later in this section: *pages*, *reference places*, and *reference transitions*, where a reference place or a reference transition is called a *reference node.*

[1]Note that the PNML meta model allows arcs between nodes of the same kind. The reason is that there are Petri net types with such arcs. Since Petri net types only restrict the meta model, the meta model should not forbid such arcs.

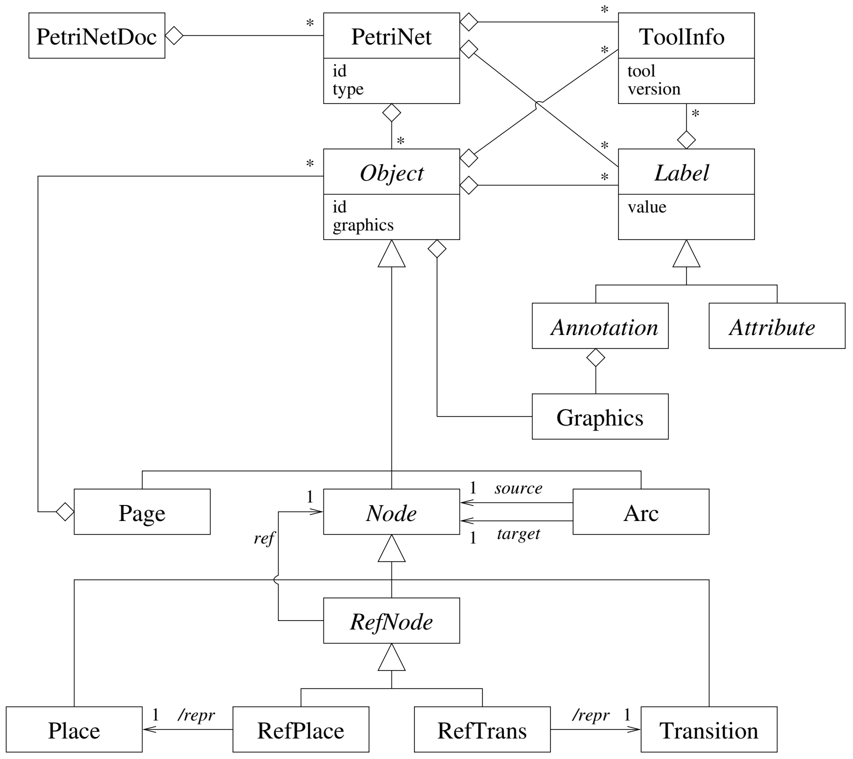

Figure 2: The PNML meta model

4.2.2 Labels.

In order to assign further meaning to an object, each object may have *labels*. Typically, a label represents the name of a node, the initial marking of a place, the guard of a transition, or the inscription of an arc. In addition, the Petri net itself may have some labels. For example, the declarations of functions and variables that are used in the arc inscriptions could be labels 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 defined by a reference to a unique *Petri Net Type Definition* (PNTD), which will be discussed in Clause. 7.
Two kinds of labels are distinguished: *annotations* and *attributes*. An annotation comprises information that is typically displayed as text near the corresponding object. Examples are names, initial markings, arc inscriptions, transition guards, and timing or stochastic information. In contrast,

the values of an attribute are chosen from a fixed and small domain. An attribute is not displayed as a text near the corresponding object. Rather, the attribute has an effect on the shape of the corresponding object. For example, an attribute *arc type* could have domain `normal`, `read`, `inhibitor`, `reset`. PNML does not define the effect on the graphical appearance of an attribute, although the Standard Features Definition Document may provide directions.

```
< actually, we could equip the Features Defintion Interface with
a mechanism for defining the graphical appearance of objects with
a particular attribute.  But, this will require some effort to make
it really work.  I don't know whether we have capacity for that.
>
```

4.2.3 Graphical information.

Each object and each annotation is equipped with graphical information. 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 annotation, it includes its relative position with respect to the corresponding object[2]. There can be additional information concerning size, colour, and shape of nodes or arcs, or concerning colour, font, and font size of labels (see Clause 5.4 for details).

4.2.4 Tool specific information.

For some tools, it might be necessary to store tool specific information, which is not supposed to be used by other tools. In order to store this information, each object and each label may be equipped with such *tool specific information.* 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.

[2]For an annotation of the net itself, the position is absolute.

4.2.5 Pages and reference nodes.

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 objects – it may even consist of other pages. An arc, however, may connect nodes on the same page only[3]. 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.
Reference nodes may have labels but these labels can only specify information about their appearance. They cannot specify any information about the referenced node itself, which already has its own labels for this purpose.

5 PNML Syntax (and Semantics)

5.1 Syntax

In this Clause, the syntax of PNML will be presented, i. e. how the concepts of PNML are mapped to XML. Here, we give an immediate translation from the concepts of PNML to XML. Appendix A, gives a formal definition of valid PNML documents in terms of a RELAX NG grammar.

5.2 PNML Meta Model

The PNML meta model is translated into XML syntax in a straightforward manner.

5.2.1 PNML elements.

Basically, each concrete class[4] of the PNML meta model is translated into an XML element. This translation along with the attributes and their data types is given in Tab. 1. These XML elements are the *keywords* of PNML and are

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

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

Table 1: Translation of the PNML meta model into PNML elements

Class	XML element	XML Attributes
PetriNetFile	`<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>`	

called *PNML elements* for short. For each PNML element, the aggregations of the meta model define in which elements it may occur as a child element. The data type ID in Tab. 1 refers to a set of unique identifiers within the PNML document. The data type IDRef refers to the set of all identifier occurring in the document, i. e. they are meant as references to identifiers. A referenc at some particular position, however, is restricted to a subset. 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[5].

5.2.2 Labels.

There are no PNML elements for labels because the meta 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 meta model (i. e. not occurring in Tab. 1) is considered as a label of the PNML element in which

[5]Note, that 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

Table 2: Elements in the `<graphics>` element depending of 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>`

it occurs. For example, an `<initialMarking>` element could be a label for 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 XML tree in a `<structure>` element[6]. An optional PNML `<graphics>` element defines its graphical appearance, and further optional PNML `<toolspecific>` elements may add tool specific information to the label.

5.2.3 Graphics.

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.
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

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

right and the y-axis from top to bottom. More details on the effect of the graphical features can be found in Sect. 5.4.1.

5.2.4 Example.

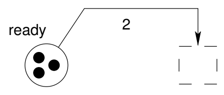

Figure 3: A simple P/T-system

In order to illustrate the structure of a PNML document, we given an example PNML document representing the Petri net shown in Fig. 3. Listing 1 shows the corresponding PNML code. 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*.

```
< it should be discussed whether we should include modular PNML into
this part already; from the feedback I (E.K.) have got, I think we
should include it
```

5.3 Semantics

```
<
- Flattening for pages / reference nodes
- No Semantics for other features
>
```

5.4 Graphical representation (non-normative)

```
< maybe, we should make this even normative?  >
```

In this Clause, we discuss the graphical features of PNML and their effect on the graphical presentation of a PNML document. Clause 5.4.1 gives an informal overview of all graphical features. Clause 5.4.2,defines the graphical presentation of a PNML document by an XSLT transformation from PNML to the *Scalable Vector Graphics* (SVG).

Listing 1: PNML code of the example net in Fig. 3

```
<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>
```

5.4.1 Graphical appearence

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.

Table 3: PNML graphical elements

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

The `<position>` element defines the absolute position of a 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.
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.
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.
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.
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.

5.4.2 Transformation to SVG

In order to give a precise defintion 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.

`< the tranformations still need some polishing; then they could go to the Appendix (non-normative) >` .

Transformations of basic PNML. The overall idea of the translation from PNML to SVG is to transform each PNML object to some SVG object,

where the attributes of the PNML element and its child elements are used to give the SVG element the intended graphical appearance.
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.
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 centre of the first line segment. If there is no `<offset>` element, the transformation uses some default value, while trying to avoid overlapping.
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.

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.
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.

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

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>
```

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.

`< type specific graphical transformations are very preliminary; I think we should not include them into this standard, because the might need much work for making this really work >`

6 Feature Definition Interface

In this Clause, we show how new labels can be defined in PNML, by the help of the Features Definition Interface.
Listing 2 shows the RELAX NG definition of the label `<initialMarking>`, which represents the initial marking of a place of a P/T-system (cf. List. 1). Its value (in a `<text>` element) should be a natural number, which is formalized by referring to the corresponding data type `nonNegativeInteger` of the data type system of XML Schema. Note that the optional graphical and tool specific information do not occur in this label definition; this is not necessary, because these standard elements for annotations are inherited from

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>
```

the standard annotation of PNML. 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.

`< a precise definition is missing yet >`

7 Type Definition Interface

Listing 3 shows the Petri Net Type Definition (PNTD) for P/T-Systems as a RELAX NG grammar. Firstly, it includes both the definitions for the meta model of PNML (`pnml.rng`) and the definitions of the Conventions Document (`conv.rng`) (Sect. ??), which, in particular, contains the definition from List. 2, a similar definition for arc inscriptions of P/T-systems, and a

definition for names.
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 and the arcs are equipped with an arc inscription. 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.

`< the precise definition of how to define a Petri type will be added later >`

8 Some Standard Types

8.1 Place/Transition Systems

`< Michael's examples >`

8.2 High-level Petri Nets

`< this is very preliminary stuff and the presentation is a little bit redundant; but it is a start >`

8.2.1 Basic idea

Here, we will briefly sketch the idea of a PNTD for HLPNGs as defined in Part 1 of this International Standard. Note that this definition is an example for having a textual as well as a structural representation of the value of a label.
The PNTD defines labels for the corresponding concepts of HLPNGs: *signatures*, *variables*, *initial markings*, *arc-inscriptions*, and *transition guards*. Arc-inscriptions and transition guards are *terms* over the signature and the variables of the HLPNG. No concrete syntax is defined, but terms are defined in the usual inductive manner, providing an abstract syntax.

`< maybe, we should also define a concrete syntax >`

In the PNTD for HLPNGs, the value of a label may be represented by using a concrete or abstract syntax. The value in a concrete syntax is represented as pure text within the label's `<text>` element, the value in an abstract syntax is represented within the label's `<structure>` element. An XML substructure within the `<structure>` element represents the inductive construction of a term from its subterms. This way, we can exchange high-level nets between two tools that use different concrete syntaxes, but have the same underlying structure (which is fixed for HLPNGs). Tools are not required to export or import abstract syntax, but if they do, interoperability with other tools is increased.

8.2.2 Concept

A HLPNG is a Petri net with the following extensions.

- A *signature* defines the domains and the functions that are used in the Petri net. In fact, it does not define the domains. It only names the *sorts* and the operations along with their arity. The sorts *bool* and *nat* are implicitly defined for each signature, so are the constants *true* and *false*.

  For each sort, the signature implicitly defines a *multiset sort* which denotes the multiset over the domain of the original sort. The operations may have the multiset sorts as parameter sorts as well as result sorts.

- A set of *variables* along with their type. From the variables and the operations of the signature, *terms* of the corresponding sorts can be built. If the sort of a term is a multiset sort, it is called *multiset term*. A term without any variables is called a *ground term*.

- Each place is equipped with a *sort*, which defines the domain of the legal tokens on this place. The marking of a place is a multiset over the corresponding domain, which can be represented as a ground term of the corresponding multiset sort. The *initial marking* of the place is represented by such a term.

- Each arc is annotated with a multiset term over the sort of the involved place. This term is often called the *arc-inscription*. The arc-inscription defines which tokens are removed from or added to the corresponding place when the transition fires.

- Each transition is equipped with a term of sort *bool*. This *transition guard* imposes additional restrictions on the firing of a transition.

In the following, we will discuss how to represent this additional information of HLPNGs by a Petri Net Type Definition.

The Petri Net Type Definition for HLPNG, basically defines a label for each of the above extensions. Note that all labels for HLPNGs are annotations. The different labels are strongly related, and, in particular, the structure of arc-inscriptions is dependent on the signature. Hence, we try to capture as much of this structure as possible within the labels. To this end, we use the feature of PNML that allows us to have structured labels as well as a textual representation of this structure. This way, it is even possible to exchange HLPNGs with a different concrete syntactical representation as long as the abstract syntax conforms to the Standard. The element `<text>` represents the textual representation of the extension in some concrete syntax. The element `<structure>` represents the structure of the extension in the fixed abstract syntax of HLPNGs.
An annotation `<definitions>` for the net in a whole defines the signature and the variables that are used in the net. The element `<text>` gives the definition in textual representation in some concrete syntax. This concrete syntax could be the concrete syntax of some programming language such as ML for Design/CPN. This textual definition could even cover aspects that are not captured in the definition of signatures at all, e.g. it could contain the definition of functions, not only their functionality. The element `<structure>`, however, represents the declaration in its abstract syntax, which must conform to the abstract syntax of signatures for HLPNGs. This abstract syntax will be sketched in Appendix B. Here, we will go though the basics labels.
It starts with an element `<declaration>`.

```
< actually, I think we should allow a signature to consist of several
declarations, which could be several global labels on different pages
>
```

Each place has an annotation `<domain>`, which defines the domain of the legal tokens on that place. Again, it consists of a textual representation within the element `<text>` in some concrete syntax and of a structural representation within an element `<structure>` in the abstract syntax of HLPNGs, a sort.

The sort must be defined in the `<declaration>`.
Each place has an annotation `<initialMarking>` too, which defines the initial marking of the place. The element `<text>` gives the initial marking in some textual representation, the element `<structure>` gives it in the abstract syntax of HLPNGs, a multiset ground term of the sort of that place: `<msgroundterm>`.
Each arc has an annotation `<inscription>`, which, again consists of a textual part `<text>` and a structural part `<structure>`. The structural part is a multiset term `<msgroundterm>` over the sort of the place.
A transition has an annotation `<guard>`, where the structural part is a boolean term: `<boolterm>`.

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. (document)

[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.

[14] Ekkart Kindler and Michael Weber. A universal module concept for Petri nets – an implementation-oriented approach. Informatik-Bericht 150, Humboldt-Universität zu Berlin, Institut für Informatik, April 2001.

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

[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. (document)

[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.

`< these references are very prelimary and the strong bias on my (E.K.) own publications should be eliminated >`

A RELAX NG Grammar for PNML (Normative)

This appendix gives a complete definition of PNML in terms of a RELAX NG grammar. We start with a definition of basic PNML and then give the extensions for structured PNML. Note that some syntactical restrictions cannot be expressed in RELAX NG. Still, these restrictions are mandadory for valid PNML documents.

A.1 RELAX NG Grammar for Basic PNML

```
<?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

```
<?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>
```

B PNTD for HLPNs

< This is just to inform you about the current state of the discussion. It is the RELAX NG grammar for high-level labels. Some text explaining the concepts can be found in Clause 8.2.2. The full and correct PNTD is still to be provided. I hope that Michael Weber can help us at this point. Fabrice Kordon, and Micheal Westergaard could help with the concepts and, maybe with concrete syntax >

```
<!-- 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 dont 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
     transtions (transition guard).
   -->

  <define name="BOOLTERM">
    <!-- I dont 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>
```