A primer on the Petri Net Markup Language and ISO/IEC 15909-2

L.M. Hillah, UPMC & CNRS UMR 7606, Paris
E. Kindler, Technical University of Denmark (DTU), Lyngby
F. Kordon, UPMC & CNRS UMR 7606, Paris
L. Petrucci, University Paris 13 & CNRS UMR 7030, Villetaneuse
N. Trèves, CNAM & Cedric, Paris

1 Introduction

In 2000, there was a workshop [1] that should foster the definition of a standard transfer format for *Petri nets* as a satellite event of the annual 'Petri Net Conference' in Aarhus. As a result of this first workshop, after many other discussions and meetings, the *Petri Net Markup Language* (*PNML*) is about to be finally adopted as ISO/IEC 15909-2. Over the years, PNML has evolved and, unfortunately, there are many different intermediate versions and variants, that are still in use. With this paper, we would like to report on the final result and on PNML as it is defined in ISO/IEC 15909-2. This way, we hope to unify the different lines of PNML and advertise the use of ISO/IEC 15909-2.

Note that this paper is not a copy or exact reproduction of ISO/IEC 15909-2 (which, including all Annexes, has more than 100 pages). Rather it is a restructured excerpt that focuses on the most important issues and abstracts from some technical details, which can be found in ISO/IEC 15909-2. Most of the technical details can be derived from the RELAX NG grammars provided at the PNML web pages [17]. Together, this should provide a fair account of the standard, its ideas and concepts, and its practical use. For a in-depth discussion of the rationales and design decisions behind PNML, we refer to the bunch of earlier publications [2, 15, 4, 18, 19]

Though not an exact copy of ISO/IEC 15909-2, this paper reuses material of ISO/IEC 15909-2 with some modifications and simplifications with the kind permission of ISO/IEC, Geneva.

Originally, PNML was introduced as an interchange format for all kinds of Petri nets [2, 3, 4]. Some major concepts of PNML were driven by this objective. Technically, ISO/IEC 15909-2 defines a transfer syntax for *High-level Petri Net Graphs* and those subclasses of *Petri nets* only that have been conceptually and mathematically defined in the International Standard ISO/IEC 15909-1 [5], for capturing the essence of all kinds of coloured and high-level Petri nets [6, 7, 8, 9, 10, 11, 12, 13]. In this paper, the focus is on PNML for high-level nets in order not to mix up concepts that are part of ISO/IEC 15909-2 and some extensions, which are currently under consideration for ISO/IEC 15909-3. In the conclusion (Sect. 5), we will briefly discuss some of these perspectives, which make PNML applicable for all kinds of *Petri nets*.

2 Concepts

In this section, we discuss the main concepts of PNML, were the main idea is that any kind of *Petri net* can be considered to be a labelled graph. In particular, all information that is specific to a particular kind of *Petri net* can be captured in *labels*.

This will be discussed in more detail in Sect. 2.1. The concepts of PNML will be defined by a meta-model for PNML in terms of UML class diagrams. These meta-models, however, do not define the XML syntax for the transfer format. Therefore, there is a mapping from the concepts of the PNML meta-model to XML, which will be discussed in Sect. 4. Curious readers, who are interested in looking at the actual XML right away, might have a look at this and, in particular, at the example in Listing 1.

As mentioned above, ISO/IEC 15909-2 defines transfer formats for different versions of *Petri nets*: *Place/Transition Nets*, *High-level Petri Net Graphs* (*HLPNG*), and *Symmetric Nets* as defined in ISO/IEC

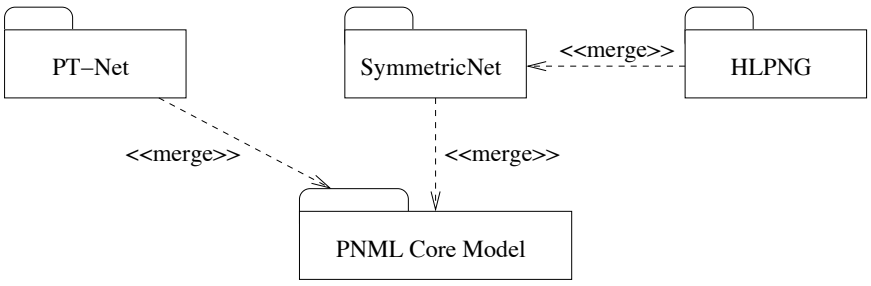

Figure 1: Overview of the UML packages of PNML

15909-1, where *Symmetric Nets* are a subclass of high-level nets, which originally were introduced under the name *well-formed nets* [14]. The main differences between them are the available data types, which will be discussed in Sect. 3.

2.1 General Principles

In order to deal with different kinds of *Petri nets*, the transfer format needs to be flexible and extensible. In order to obtain this flexibility, a *Petri net* is considered to be a labelled directed graph, where all type specific information of the net is represented in *labels*. A *label* may be associated with a *node*, an *arc*, or the *net* itself. This basic structure of a *PNML Document* is defined in the *PNML Core Model* using a UML class diagram. This will be discussed in Sect. 2.2.

The *PNML Core Model* imposes no restrictions on *labels*. Therefore, the *PNML Core Model* can represent any kind of *Petri net*. Due to this generality of the *PNML Core Model*, there can be even *PNML Documents* that do not correspond to a *Petri net* at all. For example, there could be *labels* from two different and even incompatible versions of *Petri nets* within the same *PNML Document*. For a concrete version of *Petri nets*, the legal *labels* will be defined by extending the *PNML Core Model* with another meta-model that exactly defines the legal *labels* of this particular type.

Technically, the *PNML Core Model* is a UML package, and there are additional UML packages for the different *Petri net types* that extend the *PNML Core Model* package. ISO/IEC 15909-2 defines a package for *Place/Transition Nets*, a package for *Symmetric Nets*, and a package for *High-level Petri Net Graphs*, where the package for *High-level Petri Net Graphs* extends the package for *Symmetric Nets*. Therefore, every *Symmetric Net* is also a *High-level Petri Net Graph*.

Figure 1 gives an overview of the different packages defined and on their dependencies. The package *PNML Core Model* defines the basic structure of *Petri nets*; this structure is extended by the package for each type. The *PNML Core Model* is discussed in Sect. 2.2 and the package PT-Net is discussed in Sect. 2.3.1. The general concepts of *High-level Petri Net Graphs* will be discussed in Sect. 2.3.2. The different data types used in the different versions of *high-level Petri nets* are discussed in Sect. 3.1. Based on these data types, the package SymmetricNet is discussed in Sect. 3.3.2 and the package HLPNG for general *High-level Petri Net Graph* is defined in Sect. 3.3.3.

2.2 The PNML Core Model

Figures 2 and 3 show the meta-model of the *PNML Core Model* as a UML class diagram. The diagram of Fig. 2 focusses on the conceptual parts, whereas the diagram of Fig. 3 focusses on the parts concerning the graphical representation (i. e. graphics). Note that the data type *String* is imported from a separate package *XMLSchemaDataTypes*, which is discussed in Sect. 2.2.6; since this is a technicality only, the references to this package are shown in tiny fonts in the diagram. The concepts of the *PNML Core Model* are discussed below.

2.2.1 Petri Net Documents, Petri Nets, and Objects

A document that meets the requirements of the *PNML* Core Model is called a *Petri Net Document* (PetriNetDoc) or a *PNML Document*. It contains one or more *Petri Nets* (PetriNet). Each *Petri Net* has a unique identifier and a *type*. The *type* is a name uniquely identifying a *Petri net type definition*;

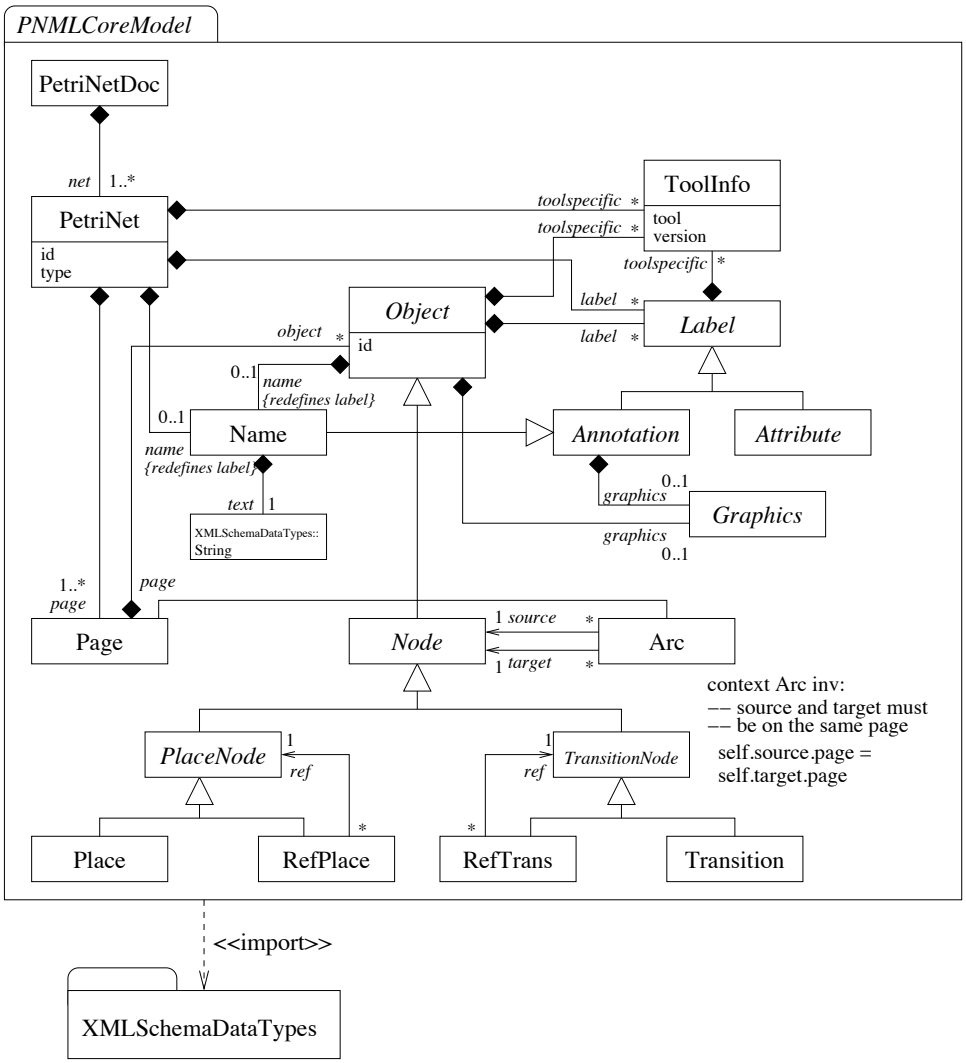

Figure 2: The *PNML Core Model* package: concepts

an example for such a name is `http://www.pnml.org/version-2009/grammar/ptnet` for the definition of *Place/Transition Nets*.

A *Petri net* consists of one or more *pages* that in turn consist of several *objects*. These *objects*, basically, represent the graph structure of the *Petri net*. Each *object* within a *Petri net document* has a unique *identifier*, which can be used for referring 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 (*graphics*). The precise graphical information that can be provided for an *object* depends on the particular type of the *object* (see Sect. 2.2.4 for more details).

The most important *objects* of a *Petri net* are *places*, *transitions*, and *arcs*. For extensibility reasons and principles of good design, *places* and *transitions* are generalised to *nodes*. For reasons explained in Sect. 2.2.2, this generalisation is via *place nodes* and via *transition nodes*. *Nodes* of a *Petri net* can be connected by *arcs*.

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

2.2.2 Pages and Reference Nodes

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

Note that PNML requires that an *arc* must connect *nodes* on the same *page* only. The reason for this requirement is that *arcs* connecting *nodes* on different *pages* cannot be drawn graphically. In the *PNML Core Model* of Fig. 2, this requirement is captured by the OCL expression next to the class for *arcs*.

In order to connect *nodes* on different *pages* by an *arc*, a representative of one of the two *nodes* must be drawn on the same *page* as the other *node*. Then, this representative may be connected with the other *node* by an *arc*. This representative is called a *reference node*, because it has a reference to the *node* it represents. Note that a *reference place* must refer to a *place* or a *reference place*, and a *reference transition* must refer to a *transition* or a *reference transition*. Moreover, cyclic references among *reference nodes* are not allowed.

2.2.3 Labels

In order to assign further meaning to an *object*, each *object* may have *labels*. Typically, there are *labels* representing the name of a *node*, the initial marking of a *place*, the *transition condition*, or some *arc annotation*. In addition, the *Petri net* itself or its *pages* may have some *labels*, which are called *global labels*. For example, the package HLPNG defines *declarations* as *global labels* of a *High-level Petri Net*, which are used for defining *variables*, and user-defined *sorts* and *operators*.

In the *PNML Core Model*, we distinguish two kinds of *labels*: *annotations* and *attributes*. An *annotation* comprises information that is typically displayed as text next to the corresponding *object*. Examples of *annotations* are names, initial markings, arc annotations, transition conditions, and timing or stochastic information. In contrast, an *attribute* is, typically, not displayed as text next to the corresponding *object*. Rather, an *attribute* has an effect on the shape or colour of the corresponding *object*. For example, an *attribute* such as arc type could have domain {`normal`, `read`, `inhibitor`, `reset`}. ISO/IEC 15909-2, however, does not mandate the effect on the graphical appearance of an *attribute*.

Note that the classes for *label*, *annotation* and *attribute* are abstract in the *PNML Core Model*, which means that the *PNML Core Model* does not define concrete *labels*, *annotations*, and *attributes*. The only concrete *label* defined in the *PNML Core Model* is the *name*, which is a *label* that can be used for any *object* within any *Petri net type*. This way, any *object* such as *nodes*, *pages*, the *net* itself, and even *arcs* can have a *name*. The value of a *name* is a *String*, which is imported from the separate package *XMLSchemaDataTypes* (see Sect. 2.2.6 for more information). All other concrete *labels* are defined in the packages for the concrete *Petri net types* (see Sect. 2.3).

In order to support the exchange of information among tools that have different textual representation for the same concepts (i. e. when they have different concrete syntax), there are two ways for representing the information within an annotation: *textually* in some concrete syntax and *structurally* as an abstract syntax tree (see Sect. 2.3.2 and 4.1.2 for details).

Note that *reference nodes* may have *labels*, but these *labels* do not have any meaning. This choice was made in order to obtain a semantically equivalent *Petri net* without *pages* by merging every *reference node* to the *node* it directly or indirectly refers to. This is called *flattening* of the *Petri net* (see [15] for details). Still, the labels of a *reference node* can have an effect on the graphical appearance or can give some additional information to the user.

2.2.4 Graphical Information

In addition to the *Petri net* concepts, information concerning the graphical appearance can be associated with each *object* and each *annotation*. For a *node*, this information includes its position; for an *arc*, it includes a list of positions that define intermediate points of the *arc*; for an *object's annotation*, it includes its relative position with respect to the corresponding *object*; for an *annotation* of a *page*, 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. Note that this information can be used for automatically transforming a *Petri Net* into *Scalable Vector Graphics* (SVG) by XSLT transformations (see [16] for more details). This transformation, however, is not part of ISO/IEC 15909-2.

Figure 3 shows the different graphical information that can be attached to the different types of *objects* and the different attributes. Note that these concepts still belong to the *PNML Core Model*; it is shown in

a different figure only in order to avoid clutter. Table 1 gives an overview of the meaning and the domain of the attributes of the different graphical features.

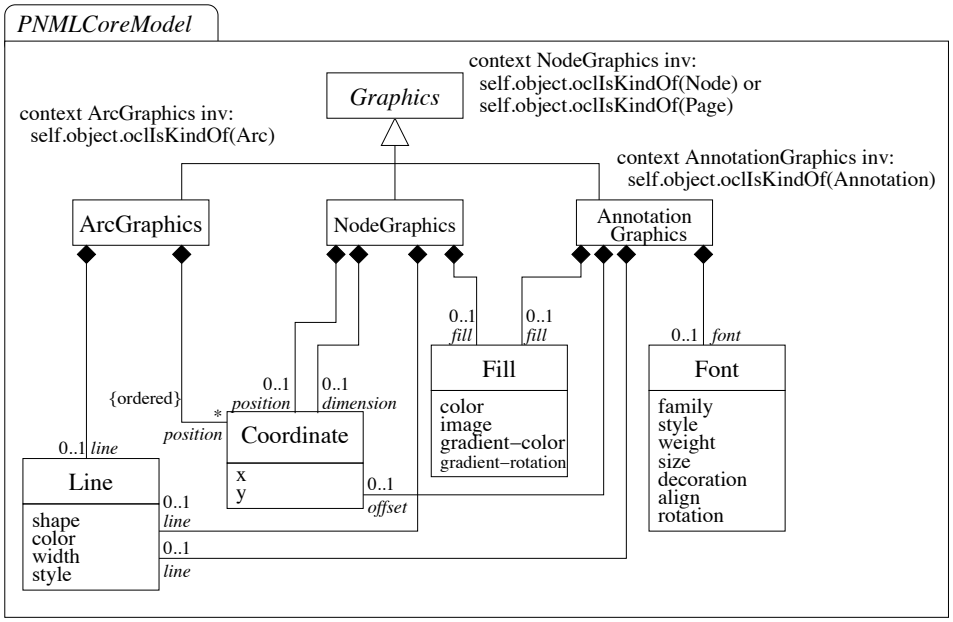

Figure 3: The *PNML Core Model*: graphical information

Table 1: PNML attributes of graphical information

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

The *position* defines the absolute position for *nodes* and *pages*. For an *annotation* the *offset* defines its relative position to the *object* it is attached to; for a *global annotation*, the *offset* defines the absolute position on that *page*. Each absolute or relative position consists of a pair of Cartesian coordinates (x, y), where the units are points (pt). As for many graphical tools, the x-axis runs from left to right and the y-axis from top to bottom. And the reference point for the position of an *object* is its centre.

For an *arc*, the (possibly empty) sequence of *positions* defines its intermediate points (bend points). Note that the positions of the *start point* and the *end point* of an *arc* are not given explicitly for the arc. These positions are determined from the position of the source and the target *nodes* of the *arc* and the direction of the first resp. last segment of the *arc*, by the intersection of this segment with the nodes border. Altogether, the *arc* is displayed as a path from the *start point* on the border of the *source node* to the *end point* on the border of the *target node* via the intermediate points. Note that, even though there

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

For *arcs*, there are also some *line* attributes, which define the *style*, the *colour*, and the *width* in which they are displayed.

Depending on the value of the attribute *shape* of element *line*, the path is displayed as a broken *line* (polyline) or as a quadratic Bezier *curve*. In the case of a Bezier curve the intermediate positions alternately are the line connectors or Bezier control points. The reference point of an arc is the middle of the arc, which is the middle of the middle segment of the arc, if there are an odd number of segments; it is the middle point, if there are an even number of segments.

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

The two elements *fill* and *line* define the interior and outline colours of the corresponding element. The value assigned to a *color* attribute must be a RGB value or a predefined colour as defined by CSS2 (Cascading Style Sheets 2). 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. When the attribute *image* is defined, the node is displayed as the image which is provided 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 attributes *family*, *style*, *weight*, and *size* are CSS2 attributes for defining the appearance of the text. The detailed description of the possible values of these attributes and their effect can be found in the CSS2 specification. The *decoration* attribute defines additional properties of the appearance of the text such as underlining, overlining, or striking-through the text. Additionally, the *align* attribute defines the alignment of the text to the *left*, *right* or *center*. The *rotation* attribute defines a clockwise rotation of the text.

The reference point of an annotation is always its centre.

2.2.5 Tool Specific Information

For some tools, it might be necessary to store *tool specific information* (ToolInfo), 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*. The internal structure of the *tool specific information* 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*.

The same *object* may be tagged with *tool specific information* from different tools. This way, the same document can be used and changed by different tools at the same time. The intention is that a tool should never change or delete the information added by another tool as long as the corresponding *object* is not deleted. Moreover, *tool specific information* should be self-contained and not refer 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. This use of the *tool specific information* is strongly recommended; however, it is not normative.

2.2.6 Data Types

In this section, we discuss six XML data types, which are taken from XMLSchema. These can be used for defining labels for some *Petri net type* with numerical and textual information. Note that these data types are not related to the data types that may be used within high-level nets! Rather, they define the types of the attributes and their XML syntax that may be used in the meta-models of the *PNML Core Model* and of the *Petri net types*.

Figure 4 gives an overview of these XML data types and their relation.

String refers to any printable sequence of characters, which is mapped to XML PCDATA. *Decimals*, *NonNegativeDecimals*, *Integer*, *NonNegativeInteger*, and *PositiveInteger* refer to any character sequence which denotes a number of the corresponding kind.

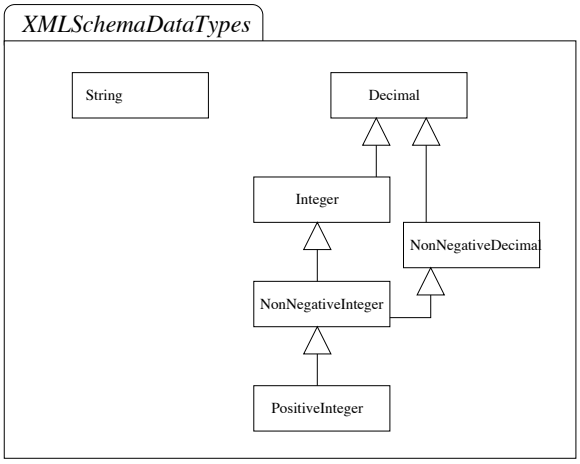

Figure 4: Standard data types

2.3 Petri Net Type Meta Models

Next, we discuss three versions of *Petri nets*: *Place/Transition Nets*, *Symmetric Nets*, and *High-Level Petri Net Graphs* (HLPNGs) as defined in ISO/IEC 15909-1, where *Symmetric Nets*[1] are a restricted version of *High-Level Petri Net Graphs* currently defined as an Amendment to ISO/IEC 15909-1. The general relation between the different types has already been shown in Fig. 1. Again, the concepts of these different versions are defined in terms of meta-models. These meta-models define the labels of the respective *Petri net type*.

ISO/IEC 15909-1 defines a mapping from the concepts of *Place/Transition Nets* to the concepts of *High-Level Petri Net Graphs* and shows how the usual mathematical representation of *Place/Transition Nets* can be represented as a restricted version of *High-level Petri Nets*. Though conceptually the same model, the syntax is different. Therefore, ISO/IEC 15909-2 introduces yet another Petri net version: *Place/Transiton Nets in High-level Notation*, which will be discussed in Sect. 3.3.1.

ISO/IEC 15909-2, however, introduces an explicit transfer format for *Place/Transition Nets* in order not to force tools for *Place/Transition Nets* to use the syntax of *High-Level Petri Net Graphs*. This format reflects the usual mathematical definition of *Place/Transition Nets*.

2.3.1 Place/Transition Nets

Since *Place/Transition Nets* are the simplest version, we start with this version: the concepts are defined in terms of a meta-model in UML notation: the package *PT-Net*.

A *Place/Transition Net* is a *net graph*, where each *place* can be labelled with a natural number representing the *initial marking* and each *arc* can be labelled with a non-zero natural number representing the *arc annotation*, with the usual meaning: the *label* of a *place* p denotes the initial marking $M(p)$, the *label* of an *arc* f denotes the arc weight $W(f)$.

Figure 5 shows the package PT-Net. Note that the only classes defined here are *PTMarking* and *PTArcAnnotation*. The classes *Place*, *Arc*, and *Annotation* come from the package *PNML Core Model*. They are imported (actually, they are merged) here in order to define the possible *labels* for the particular *nodes* of *Place/Transition Nets*. Likewise, the classes prefixed with *XMLSchemaDataTypes* are imported from the standard data type package (see Sect. 2.2.6).

The *initial marking* of a *place* is represented by the *annotation PTMarking*, the contents of which must be a non-negative integer. Technically, the representation of the contents of this *label* is defined by referring to the data type *NonNegativeInteger*.

The *arc annotation* is represented by the *annotation PTArcAnnotation*, the contents of which must be a non-zero natural number, which is defined by referring to the data type *PositiveInteger*.

Note that, according to this definition, it is legal that a *place* does not have an *annotation* for the *initial marking*. In that case, the *initial marking* is assumed to be empty, i. e. 0. Likewise, there may be no *annotation* for an *arc*. In that case, the *arc annotation* is assumed to be 1.

[1] Remember that *Symmetric Nets* were originally introduced under the name *well-formed nets* [14].

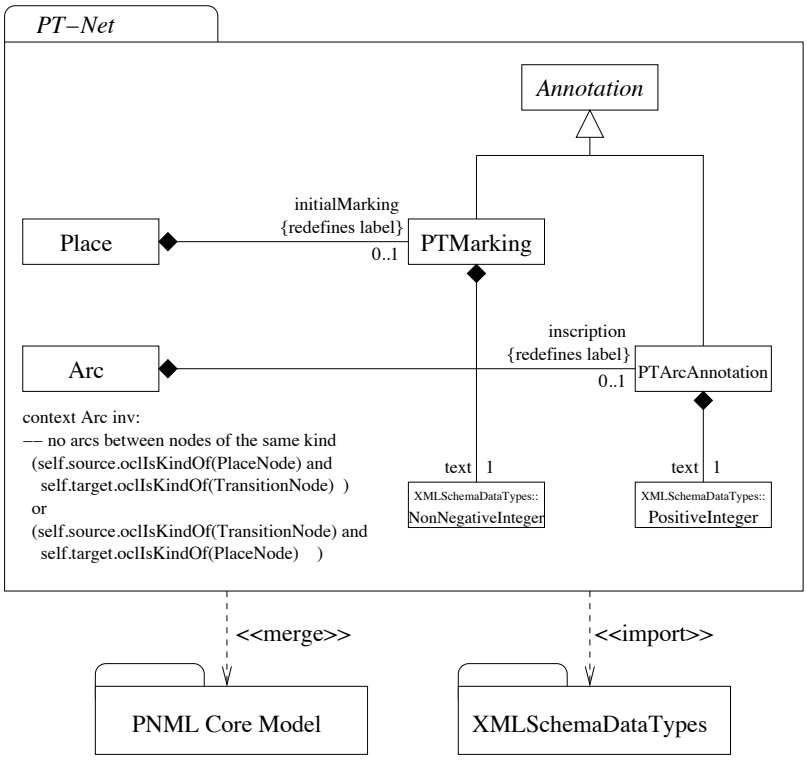

Figure 5: The package PT-Net

In addition to the definition of the new *labels*, this package also defines the structural restriction that, in a *Place/Transition Net*, an *arc* must not connect a *place* to a *place* or a *transition* to a *transition*. This is captured by the OCL expression below class *arc*.

Sometimes, one wants to store the position of the individual tokens within a place. In order not to mandate all tools to support this feature, ISO/IEC 15909-2 suggests a *tool specific information* for this purpose, which would refer to the tool *org.pnml.tool*. Since no tool is required to support tool specific features, every tool is free to use and support this feature or not.

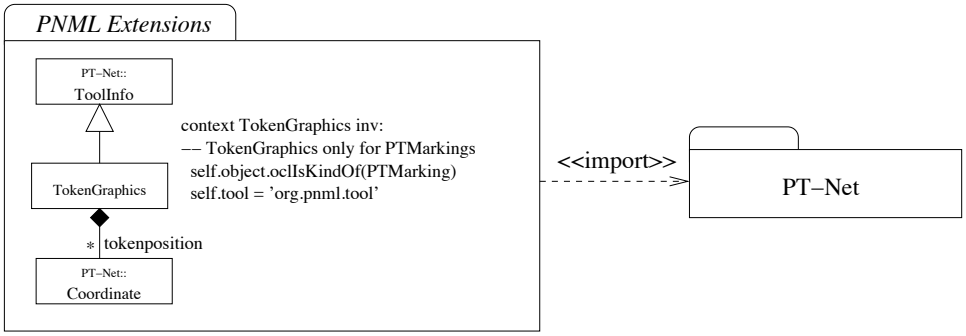

Figure 6: Tool specific extension for token positions

This extension is shown in Fig. 6. For a *PTMarking*, this label can be equipped with the *tool specific information* attached to *AnnotationGraphics*. This consists of information on a list of *tokenpositions*. Each of these elements represents the position of a token relative to the centre of the *place*; represented as a *Coordinate*, which is imported from package *PT-Net*, where it was obtained by a merge with the *PNML Core Model*. If this graphical information is present at all, the number of *tokenpositions* should be the same as indicated by the *PTMarking* (see example in Sect. 4.1.5).

2.3.2 High-Level Core Structure

ISO/IEC 15909-1 defines *High-level Petri Net Graphs*. *Symmetric Nets* and *Place/Transition Nets* are introduced as restricted versions of *High-level Petri Net Graphs*. The difference is in the *types* and *functions* that may be used in the different versions. To cater for this structure, we distill the common structure of all *High-level Petri Net Graphs* first: This is called the *High-Level Core Structure*. The built-in data types of the specific Petri net types are discussed in Sect. 3.1.

Syntactically, the basic features of a *High-Level Petri Net* are the *annotations* of *places*, *transitions*, and *arcs*. For each *place*, a *sort* defines the *type* of the tokens on this *place*. A *term* associated with a *place* denotes the *initial marking* and must have the respective *sort*. The *term* associated with an *arc* to or from a *place*, defines which tokens are added or removed, when the corresponding *transition* fires. These *terms* must also be of the respective *sort*.

For constructing such *terms*, one can use built-in *operators* and *sorts*, and user-defined *variables*, which are defined in a *variable declaration*. The *variable declarations* are *annotations* of the *net* or a *page*. Moreover, a *transition* can have a *condition* which is a *term* of *sort* boolean and imposes additional conditions on the situations in which a transition can fire.

The meta-model for *terms*, which defines all these concepts, is shown in Fig. 7: It defines the concepts of *sorts*, *operators*, *declarations*, and *terms*, and how *terms* are constructed from *variables* and *operators*.

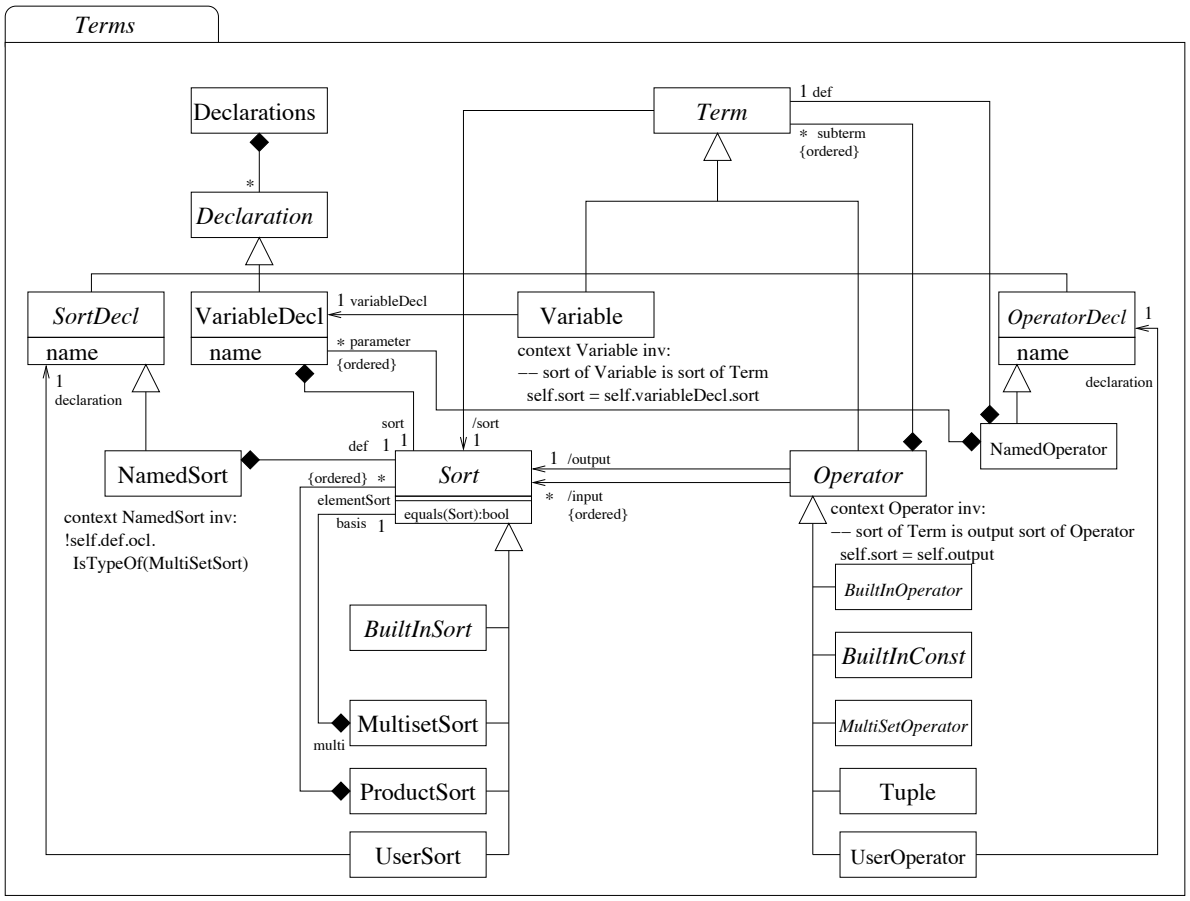

Figure 7: The meta-model for *Terms*

For each *variable declaration*, there is a corresponding *sort*. A *sort* can be a *built-in sort*, a *multiset sort* over some *basis sort*, a *product sort* over some *sorts*, or a *sort* which is given in a *user declaration*. In the core structure, the only possible *sort declaration* of a user is by constructing a new *sort* from existing ones and by giving them a new name. From these, the user can define new ones. Note that cyclic references in user-defined *sorts* are not allowed. In addition to these user-defined sorts, called *named sorts*, there will be *arbitrary sorts*. Since these are not allowed in *Symmetric Nets*, these are not defined in the core. They are discussed in Sect. 3.3.3 where the concepts of general high-level nets are explained.

An *operator* can be a *built-in constant* or a *built-in operator*, a *multiset operator* which among others can construct a *multiset* from an enumeration of its elements, or a *tuple* operator. Each *operator* has a sequence of *sorts* as its *input sorts*, and exactly one *output sort*, which defines its signature. As for *sorts*, the user can define his own *operators*. Here, it is only possible to define an abbreviation, which is called a *named operator*: It can use a *term*, which is built from existing *operators* and parameter variables, for defining a new *operator*. As for *sorts*, cycles (recursion) in these definitions are not allowed. As for sorts, there will be arbitrary operator declarations for *High-level Petri Net Graphs*, but not for *Symmetric Nets*. Therefore, these concepts will be discussed in Sect. 3.3.3.

From the built-in *operators* and the user-defined *operators* and *variables*, *terms* can be constructed in the usual way. The *sort* of a *term* is the *sort* of the *variable* or the *output sort* of the *operator*. Therefore, *sort* is indicated as a derived association; its definition is expressed by an OCL expression in the UML metamodel. Note that the *input* and *output sorts* of the *operator* are also represented as derived associations because, in some situations, they need not be given explicitly since they can be derived from the type of the *operator*.

Figure 8 shows the package *High-Level Core Structure*, which defines all the *annotations* for both *Symmetric Nets* and *High-Level Petri Net Graphs*. Note that, since the classes for built-in *sorts* and *operators* are abstract, we do not have any built-in *sorts* and *operators* yet. These are discussed in Sect. 3.1.

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

Note that this model defines an abstract syntax (composition *structure*) for all these concepts only. In order to allow tools to store the concrete text, all *annotations* of *High-level Nets* may also consist of text, which should be the same expression in the concrete syntax of some tool. This concrete syntax, however, is not mandated by ISO/IEC 15909-2; therefore, all the meaning is in the *structure* of the labels!

3 Details of High-level nets and their data types

As mentioned earlier, Sect. 2.3.2 discussed the core structure of *High-level Petri Net Graphs* only. In this section, we briefly discuss the details of the built-in *sorts* and *operators*. For each built-in *sort*, there is a UML package. Sections 3.1 gives an overview of these packages. Section 3.2 briefly discusses the package for user-defined declarations. Based on that, Sect. 3.3 defines the different types of high-level nets.

3.1 Data types

ISO/IEC 15909-2 defines the following data types for being used in the different versions of high-level Petri nets:

- *Dots* defines the sort which represents a type with exactly one element (token): $\{\bullet\}$. This is used to represent *Place/Transition Nets* as *High-level Net Graphs*.
- *Multiset* defines the multiset over any other sort, which are needed for defining the terms labelling arcs and for initial markings.
- *Booleans* defines the boolean type and associated operations.
- *Finite Enumerations*, *Cyclic Enumerations*, and *Finite Integer Ranges* allow the definition of finite ranges by explicitly enumerating them or by giving an integer range. Depending on the type, different structures are imposed on them [14], such as an order relation, or a successor and predecessor operation.
- *Partitions* allow the definition of finite enumerations that are partitioned into sub-ranges, where a partition function defines for each element to which partition it belongs. These partitions define a separate sort, where each partition is an element of the respective type.
- *Integer* defines the integers and the usual operations on them; it also defines the subtypes positive numbers and the natural numbers.
- *Strings* defines the strings and the usual operations on them.

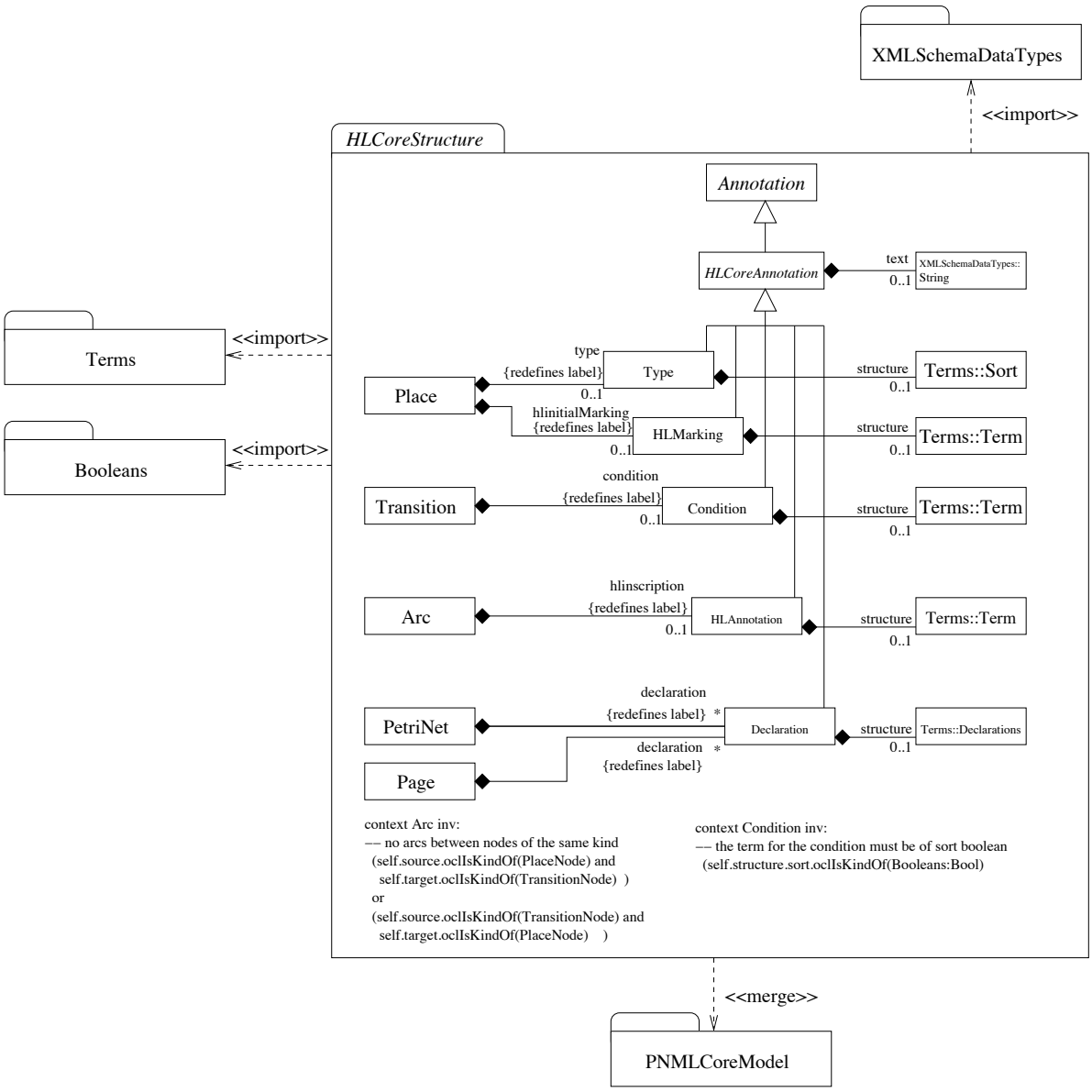

Figure 8: The package *High-Level Core Structure*

- *Lists* defines the Lists over the other data types and the usual operations on them.

For details, we refer to ISO/IEC 15909-2 or to the RELAX NG grammars which can be found at [17].

3.2 User declarations

In general *High-Level Petri Net Graphs* (HLPNGs), the user is allowed to define arbitrary *sorts* and *operators*. This package *ArbitraryDeclarations* is shown in Fig. 9. In contrast to *named sorts* and *named operators*, arbitrary *sorts* and *operators* do not come with a definition of the *sort* or *operation*; they just introduce a new symbol without giving a definition for it. So, these symbols do not have a meaning, but can be used for constructing *terms*.

The additional concept *Unparsed* in *terms* provides a means to include any text, which will not be parsed and interpreted by the tools. This is helpful for exchanging the general structure of a term, but not all its details.

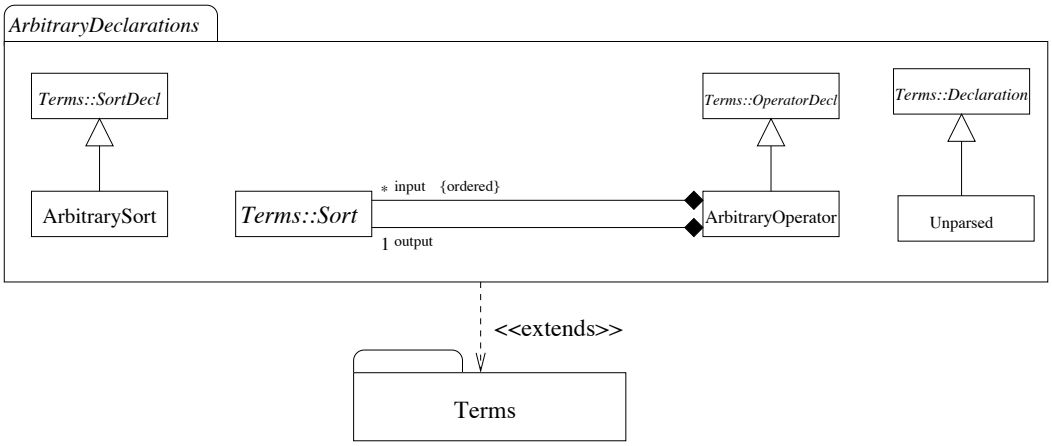

Figure 9: The package *ArbitraryDeclarations*

3.3 Net types

Based on the data types introduced in the previous section, we can now discuss the different versions of high-level nets. Note that, conceptually, high-level nets extend *Place/Transition-Systems*. But, syntactically they do not. That is why all these types are not extending the package *PT-Net*, which is discussed in Sect. 2.3.1.

Instead, the different versions of high-level nets form a separate hierarchy (see also Fig. 1). The lowest level in this hierarchy are *Place/Transition Nets in High-level Notation*.

3.3.1 Place/Transition Nets as High-level Net Graphs

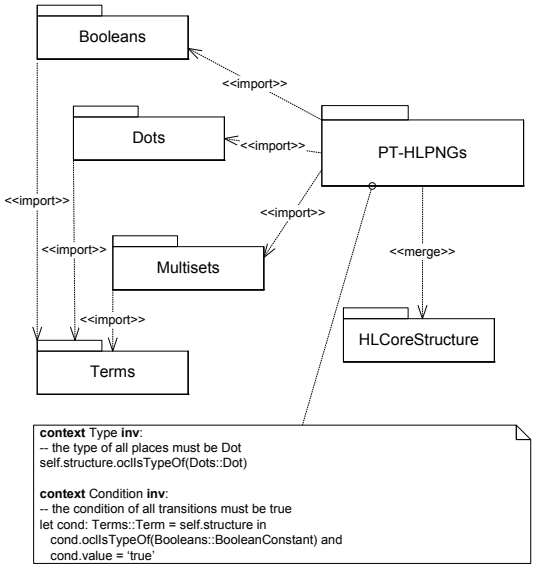

Figure 10: The package of *P/T Nets* defined as restricted *HLPNGs*

Annex B of ISO/IEC 15909-1 defines *Place/Transition Nets* as a restricted form of *High-level Net Graphs*. Fig. 10 shows the corresponding UML definition. This class allows for the use of the built-in *sorts Bool* and *Dot* only. It does neither allow any user *declarations*, nor *variables*, nor *sorts*, nor *operators*.

The *type* of each *place* must refer to *sort Dot*. All transition *conditions* need to be the constant *true*, if this label is present. And the *arc annotations* and the *initial markings* are *ground terms* of the *mulitset sort* over *Dot*.

3.3.2 Symmetric Nets

Symmetric Nets as currently defined in an Amendment (Annex B.2) of Part 1 of ISO/IEC 15909, are *High-Level Petri Net Graphs* with some restrictions. Basically, the carrier sets of all basic *sorts* are finite, and only a fixed set of *operations*, as defined below, are allowed. Therefore, the *sort* of a *place* must not be a *multiset sort*. The available built-in *sorts* are defined below.

Moreover, for *Symmetric Nets*, it is required that every HLPNG *annotation* has the *structural* information.

The built-in *sorts* of *Symmetric Nets* are the following: *Booleans*, *range of integers*, *finite enumerations*, *cyclic enumerations* and *dots*. Moreover, for every *sort*, there is the operator *all*, which is a multiset that contains exactly one element of its basis *sort*. This is often called the broadcast function.

Altogether, the *Symmetric Nets* can be defined by importing the packages as shown in Fig. 11.

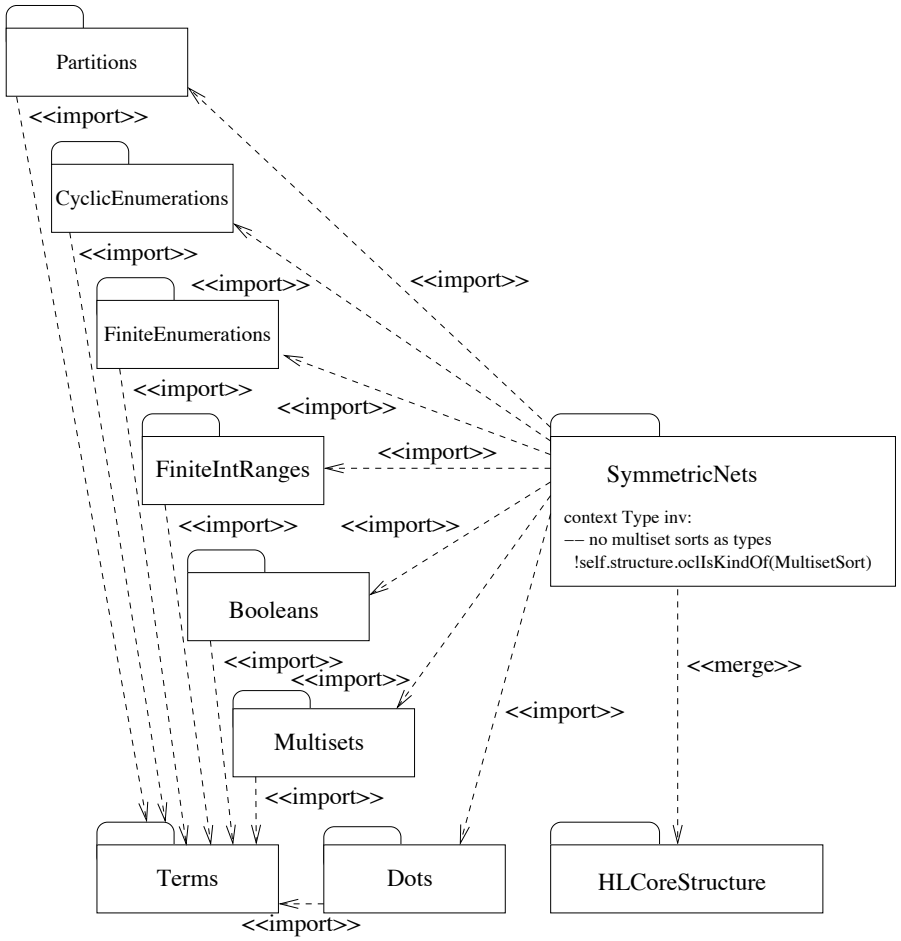

Figure 11: The package *Symmetric Nets* and its built-in *sorts* and *functions*

3.3.3 High-level Petri Net Graphs

The complete definition for *High-Level Petri Net Graphs* is shown in Fig. 12. It extends *Symmetric Nets* by declarations for *sorts* and *functions* and the additional built-in *sorts* for *Integer*, *String*, and *List*.

4 XML Syntax

Section 2 and 3 discussed the concepts of the *PNML Core Model* and the concepts of *Place/Transition Nets*, *High-level Petri Net Graphs*, and *Symmetric Nets* in terms of UML meta-models (and some additional constraints). This, however, does not define the concrete XML syntax for representing these concepts.

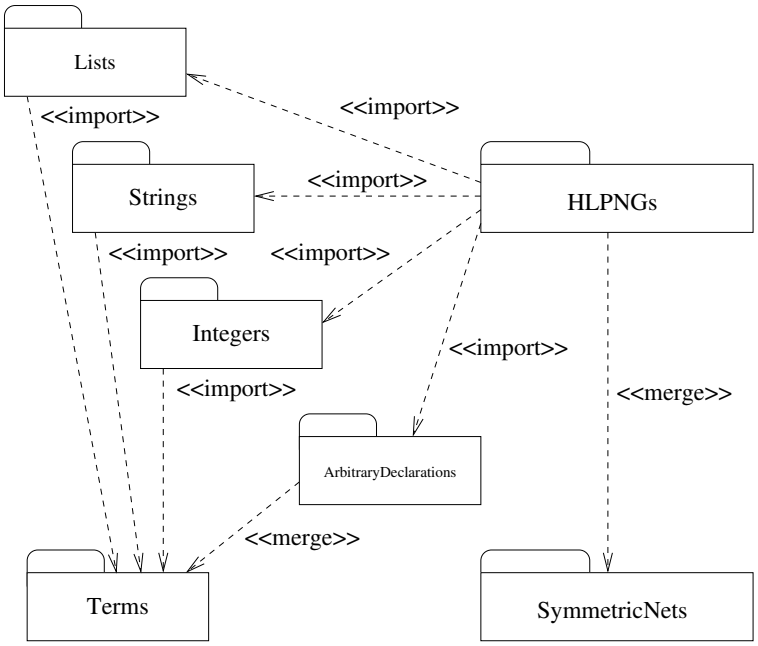

Figure 12: The package *HLPNGs*

The XML syntax of the *PNML Core Model* and the three *Petri net types* is discussed in this section. A RELAX NG grammar defining the exact XML syntax can also be found at the PNML web site [17].

In Sect. 4.1, we discuss the general format of *PNML Documents*; i. e. the XML syntax for the *PNML Core Model*. In Sect. 4.2, we discuss the format for *Place/Transition Nets*. In fact, for simple *Petri net types* such as *Place/Transition Nets* there are some general rules, as to how the concepts of a package defining some *Petri net type* are mapped to XML syntax. This idea is discussed here by the help of the example of *Place/Transition Nets*. For *High-level Petri Nets*, however, there is a dedicated mapping for the *labels* of these types to XML. This is discussed in Sect. 4.3.

Note that there is no need for a separate mapping for *Symmetric Nets* and *Place/Transition Nets in High-level Notation* to XML since, as a restricted version of *High-level Petri Nets*, these mappings are fully covered by the mapping for *High-level Petri Nets*.

4.1 XML syntax of the PNML Core Model

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

4.1.1 PNML Elements

Each concrete class[2] of the *PNML Core Model* is mapped to an XML element. The mapping of these classes along with the attributes and their data types is given in Table 2. These XML elements are the *keywords* of PNML and are called *PNML elements* for short. For each *PNML element*, the compositions of the *PNML Core Model* define in which elements it may occur as a child element.

The data type ID in Table 2 refers to a set of unique identifiers within the *PNML Document*. The data type IDRef refers to the set of all identifiers occurring in the document, i. e. they are meant as references to identifiers. A reference at some particular position, however, is restricted to objects of a particular type — as defined by the resp. associations in the *PNML Core Model*. For instance, the attribute `ref` of a *reference place* must refer to a *place* or a *reference place* of the same *net*. The set to which a reference is restricted is indicated in the table (e. g. for a *reference place*, the attribute `ref` should refer to the `id` of a *Place* or a *RefPlace*). Note that these requirements are defined in the UML meta-model already; here, these requirements are repeated just for better readability.

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

Table 2: Translation of the PNML Core Model into PNML elements

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

Note that the <pnml> element must have a namespace attribute xmlns. For the current version of PNML, this namespace is fixed: http://www.pnml.org/version-2009/grammar/pnml

4.1.2 Labels

Except for *names*, there are no explicit definitions of *PNML elements* for *labels* because the *PNML Core Model* does not define other *labels*. For concrete *Petri net types*, such as *Place/Transition Nets*, *Symmetric Nets*, and *High-level Petri Nets* the corresponding packages define these *labels*.

In general *PNML Documents*, any XML element that is not defined in the *PNML Core Model* (i.e. not occurring in Table 2) 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, and <inscription>, which represents an arc annotation.

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

An optional PNML <graphics> element defines its graphical appearance; and optional PNML <toolspecific> elements may add tool specific information to the label. Note that ISO/IEC 15909-2 does not mandate the inner structure of <toolspecific> elements; every tool is free to structure its information inside that element at its discretion; as long as it produces well-formed XML.

4.1.3 Graphics

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

Table 4 explicitly list the attributes for each graphical element defined in Table 3 (cf. Fig. 3 and Table 1). The domain of the attributes refers to the data types of either XML Schema, or Cascading Stylesheets 2 (CSS2), or is given by an explicit enumeration of the legal values.

4.1.4 Mapping of XMLSchemaDataTypes concepts

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

Table 3: Possible child elements of the <graphics> element

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

Table 4: 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

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

4.1.5 Example

In order to illustrate the structure of a *PNML Document*, there is a simple example *PNML Document* representing the *Petri net* shown in Fig. 13, which is a *Place/Transition Net*. Listing 1 shows the corresponding *PNML Document* in XML syntax. It is a straightforward translation, where there are *labels* for the *names* of objects, for the *initial markings*, and for *arc annotations*.

Note that, in this figure, the *initial marking* is not displayed as a textual *label* – it is shown graphically. This is due to the fact that the *initial marking* comes with a *tool specific* information on the positions of the *tokens*, tokens are shown at the individual positions as given in the elements <tokenposition>.

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

4.2 XML syntax of the Petri net types

Based on the *PNML Core Model* of Sect. 2.2, Sect. 2.3.1 and 3.3 discussed the definition of some *Petri net types*, which restrict *PNML Documents* to the particular *labels* defined in the corresponding packages.

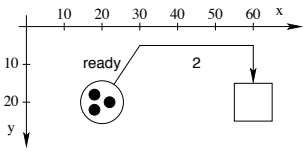

Figure 13: A simple *Place/Transition Net*

Listing 1: PNML code of the example net in Fig. 13

```
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
  <net id="n1" type="http://www.pnml.org/version-2009/grammar/ptnet">
   <page id="top-level">
    <name>
      <text>An example P/T-net</text>
    </name>
    <place id="p1">
      <graphics>
        <position x="20" y="20"/>
      </graphics>
      <name>
        <text>ready</text>
        <graphics>
          <offset x="0" y="-10"/>
        </graphics>
      </name>
      <initialMarking>
        <text>3</text>
        <toolspecific tool="org.pnml.tool" version="1.0">
          <tokengraphics>
            <tokenposition x="-2" y="-2" />
            <tokenposition x="2" y="0" />
            <tokenposition x="-2" y="2" />
          </tokengraphics>
        </toolspecific>
      </initialMarking>
    </place>
    <transition id="t1">
      <graphics>
        <position x="60" y="20"/>
      </graphics>
    </transition>
    <arc id="a1" source="p1" target="t1">
      <graphics>
        <position x="30" y="5"/>
        <position x="60" y="5"/>
      </graphics>
      <inscription>
        <text>2</text>
        <graphics>
          <offset x="0" y="5"/>
        </graphics>
      </inscription>
    </arc>
   </page>
  </net>
</pnml>
```

Table 5: Tool specific information for token positions

XML element	Attribute	Domain
`<tokengraphics>`		
`<tokenposition>`	x	decimal
	y	decimal

These packages define the *labels* that are used in the particular *Petri net*. Here, it is shown how to map such a package to the corresponding XML syntax. This mapping is the same for all type definitions, unless the type definition explicitly defines a dedicated mapping for this type. This general mapping will be explained by the help of the example of *Place/Transition Nets* (see Fig. 5 and the *PNML Document* in Listing 1).

The PT-Net package defines two kinds of *labels* that can be used in a *Place/Transition Net*: PTMarkings and PTAnnotations. Each *place* can have one *annotation* PTMarking, and each *arc* can have one *annotation* PTAnnotation. This is indicated by the compositions in the UML diagram in Fig. 5.

The XML syntax for these *labels* is derived from the role names of these compositions. Every *annotation* PTMarking is mapped to an XML element `<initialMarking>`, and every *annotation* PTAnnotation is mapped to an element `<inscription>`. Listing 1 shows an example for the XML syntax of a *Place/Transition Net*.

Since all *labels* in this package are *annotations*, all graphical elements defined for *annotations* may occur as children in these elements.

In the PT-Net package each *label* is defined to have a `<text>` element, which defines the actual content of this *annotation*. For *Place/Transition Nets* there are no structured elements.

The corresponding classes from package DataTypes define the XML content of the `<text>` element. These were discussed in Sect. 4.1.4 already.

Note that for *Place/Transition Nets*, there is a predefined tool specific extension for the label `<initialMarking>` (see Sect. 2.3.1), which represents the positions of tokens within a place. The class *TokenGraphics* is mapped to the XML element `<tokengraphics>` with no attributes. The token positions contained by this element are represented by the elements `<tokenposition>` as shown in Table 5. Within the element `<tokengraphics>` there can be any number of `<tokenposition>` elements with two attributes x and y.

4.3 Mapping for *High-Level Nets*

Table 6 defines the mappings between the meta-model elements and their XML representation. In these tables, only meta-model elements that have corresponding PNML elements or attributes are displayed. Thus, most abstract elements are not shown in the tables, except if they have attributes. All attribute types are mapped to *XMLSchema-datatypes* library data types. For details, we refer to the RELAX NG grammars [17].

5 Conclusion

In this paper, we have outlined the main concepts and the syntax of PNML as defined in ISO/IEC-15909-2, which passed its final ballot and will hopefully be published soon. For lack of space, we could not discuss all the details. In particular, we could not discuss the rationale behind the design of PNML and we could not discuss all built-in data types and their XML syntax. For the design rationale, we refer to earlier publications [2, 15, 4] and for some more information on the XML syntax we refer to the PNML web pages [17], which has the full RELAX NG grammar for all currently standardized Petri net types. The ultimate source of information will be the International Standard ISO/IEC 15909-2.

Another great help for implementing the PNML might be the PNML Framework [18], which provides an API for reading and writing Petri nets in PNML. This API is automatically generated from the PNML meta-models in the *Eclipse Modeling Framework* (EMF). More details can be found on the PNML web pages [17].

These web pages are also used for discussing future extensions, some of which might be covered in Part 3 of ISO/IEC 15909. This concerns in particular the precise definition of how to define a *Petri*

Table 6: *High-level meta-model elements and their PNML constructs*

Model element	**PNML element**	**PNML attributes**
Booleans::Bool	bool	
Booleans::And	and	
Booleans::Or	or	
Booleans::Not	not	
Booleans::Imply	imply	
Booleans::Equality	equality	
Booleans::Inequality	inequality	
Booleans::BooleanConstant	booleanconstant	value: boolean
HLCoreStructure::Declaration	declaration	
HLCoreStructure::Type	type	
HLCoreStructure::HLMarking	hlinitialmarking	
HLCoreStructure::Condition	condition	
HLCoreStructure::HLAnnotation	hlinscription	
Terms::Declarations	declarations	
Terms::VariableDeclaration	variabledecl	id: ID; name: string
Terms::OperatorDeclaration	No element	id: ID; name: string
Terms::SortDeclaration	No element	id: ID; name: string
Terms::Variable	variable	variabledecl: IDREF
Terms::NamedSort	namedsort	
Terms::NamedOperator	namedoperator	
Terms::Term	No element	
Terms::Sort	No element	
Terms::MultisetSort	multisetsort	
Terms::ProductSort	productsort	
Terms::UserSort	usersort	declaration: IDREF
Terms::Tuple	tuple	
Terms::Operator	No element	
Terms::UserOperator	useroperator	declaration: IDREF
ArbitraryDeclarations::ArbitrarySort	arbitrarysort	
ArbitraryDeclarations::Unparsed	unparsed	
ArbitraryDeclarations::ArbitraryOperator	arbitraryoperator	

net type (which where used informally only up to now) and its *features*, and the use this *Petri net type definition* mechanism for defining some more standard *Petri types* such as timed and stochastic Petri nets. Another major issue is the definition of a module concept for PNML [15, 19, 20]. This setting should be powerful enough to encompass most structuring mechanisms. Hence, it will be possible to exchange modular and hierarchical models, using different structuring paradigms, between tools so as to apply their specific analysis techniques.

If you are interested in some of these issues, you might also want to join the work of WG19 of ISO/IEC JTC1 SC7 or the discussion list available at the PNML web pages. All ideas are welcome.

Acknowledgements The work on PNML and ISO/IEC 15909-2 has been going on over 10 years now and many people have been involved in the discussion and development process in different stages and different intensity. Since, it is hard to weigh the individual contributions, we thank all of them in alphabetical order: Joao Paulo Barros, Jean Bérubé, Jonathan Billington, Didier Buchs, Søren Christensen, Jörg Desel, Erik Fischer, Giuliana Franceschinis, Guillaume Giffo, Jun Ginbayashi, Kees van Hee, Nisse Husberg, Kurt Jensen, Matthias Jüngel, Albert Koelmans, Olaf Kummer, Kjeld Høyer Mortensen, Reinier Post, Wolfgang Reisig, Stefan Roch, Karsten Wolf, Christian Stehno, Kimmo Varpaaniemi, Michael Weber, Lisa Wells, Jan Martijn van der Werf, and Lukasz Zogolowek.

We would also like to thank ISO/IEC for their kind permission to publish this paper based on material of the upcoming standard ISO/IEC 15909.

References

[1] Bastide, R., Billington, J., Kindler, E., Kordon, F., Mortensen, K.H., eds.: Meeting on XML/SGML based Interchange Formats for Petri Nets, University of Aarhus, Dept. of Computer Science (2000)

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

[3] Weber, M., Kindler, E.: The Petri Net Kernel. In Ehrig, H., Reisig, W., Rozenberg, G., Weber, H., eds.: Petri Net Technologies for Modeling Communication Based Systems. Volume 2472 of LNCS. Springer (2003) 109–123

[4] Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, technology, and tools. In van der Aalst, W., Best, E., eds.: Application and Theory of Petri Nets 2003, 24^{th} International Conference. Volume 2679 of LNCS., Springer (2003) 483–505

[5] ISO/IEC: Software and Systems Engineering – High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, International Standard ISO/IEC 15909 (2004)

[6] Genrich, H.J., Lautenbach, K.: System modelling with high-level Petri nets. Theoretical Computer Science **13** (1981) 109–136

[7] Jensen, K.: Coloured Petri nets and invariant methods. Theoretical Computer Science **14** (1981) 317–336

[8] Berthomieu, B., Choquet, N., Colin, C., Loyer, B., Martin, J., Mauboussin, A.: Abstract Data Nets combining Petri nets and abstract data types for high level specification of distributed systems. In: Proceedings of VII European Workshop on Application and Theory of Petri Nets. (1986)

[9] Vautherin, J.: Parallel systems specifications with coloured Petri nets and algebraic specifications. In Rozenberg, G., ed.: Advances in Petri Nets. Volume 266 of LNCS. Springer-Verlag (1987) 293–308

[10] Billington, J.: Many-sorted high-level nets. In: Proceedings of the 3rd International Workshop on Petri Nets and Performance Models, IEEE Computer Society Press (1989) 166–179

[11] Reisig, W.: Petri nets and algebraic specifications. Theoretical Computer Science **80** (1991) 1–34

[12] Jensen, K., (Eds.), G.R.: High-level Petri Nets, Theory and Application. Springer-Verlag (1991)

[13] Jensen, K.: Coloured Petri Nets, Volume 1: Basic Concepts. EATCS Monographs on Theoretical Computer Science. Springer-Verlag (1992)

[14] Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well-formed coloured nets and their symbolic reachability graph. In Jensen, K., Rozenberg, G., eds.: Petri Nets: Theory and Application. (1991) 373–396

[15] Weber, M., Kindler, E.: The Petri Net Markup Language. In Ehrig, H., Reisig, W., Rozenberg, G., Weber, H., eds.: Petri Net Technologies for Modeling Communication Based Systems. Volume 2472 of LNCS. Springer (2003) 124–144

[16] Stehno, C.: Petri Net Markup Language: Implementation and Application. In Desel, J., Weske, M., eds.: Promise 2002. Lecture Notes in Informatics P-21., Gesellschaft für Informatik (2002) 18–30

[17] PNML team: PNML.org: The Petri Net Markup Language home page. (URL `http://www.pnml.org/`) 2009/06/8.

[18] Hillah, L., Kordon, F., Petrucci, L., Trèves, N.: Model engineering on Petri nets for ISO/IEC 15909-2: API framework for Petri net types metamodels. Petri Net Newsletter **69** (2005) 22–40

[19] Kindler, E.: Modular PNML revisited: Some ideas for strict typing. In: Proc. AWPN 2007, Koblenz, Germany. (2007)

[20] Kindler, E., Petrucci, L.: Towards a standard for modular Petri nets: A formalisation. In Franceschinis, G., Wolf, K., eds.: Application and Theory of Petri Nets 2009, Internat. Conference, Proceedings. Volume 5606 of LNCS., Springer-Verlag (2009) 43–62