# PN Standardisation: A Survey

L. Hillah[1], F. Kordon[1], L. Petrucci[2], and N. Trèves[3]

[1] Université P. & M. Curie - Paris 6, CNRS UMR 7606 - LIP6/MoVe
4, place Jussieu, F-75252 Paris CEDEX 05, France
Fabrice.Kordon@lip6.fr, Lom-Messan.Hillah@lip6.fr
[2] LIPN, CNRS UMR 7030, Université Paris XIII
99, avenue Jean-Baptiste Clément
F-93430 Villetaneuse, France
Laure.Petrucci@lipn.univ-paris13.fr
[3] Cedric, CNAM
292, rue St Martin
F-75141 Paris Cedex 03, France
treves@cnam.fr

**Abstract.** Petri Nets formalism requires standardisation to facilitate the work of researchers in this field and to enable the data exchange between different Petri Nets tools through a common format. Following this, a three-part International Standard (ISO/IEC 15909) has been developed. Part 1 is devoted to terms and definitions for Place/Transition Nets and High-Level Petri Nets. It is now completed (published as a standard) but will include an addendum on Symmetric Nets. Part 2 aims at providing a transfer format for High-level Petri Nets, called PNML, based on XML. Work on part 3 which deals with extensions has not started yet. In this paper the first two parts of the standard are presented. Then, to support part 2, an implementation of PNML, through an API framework to be integrated into Petri Net tools, is proposed. It allows for the translation of any Petri Net, designed by a given tool in a dedicated format, into PNML.

## 1 The Challenge of PN Standardisation

Petri Nets [4,8,26,28] are a mathematically defined formalism and may thus be used to provide unambiguous specifications and descriptions of applications. They are especially dedicated to specify and design discrete event systems and this technique is particularly suited to parallel and distributed systems development as it supports concurrency. The technique allows for specification of systems at a level which is independent of the implementation choices (i.e., by software, hardware — electronic and/or mechanical — or humans, or a combination of these) and has been widely used to describe telecommunication systems, protocols, microprocessor architectures,... since their invention in 1962. They also constitute an executable technique, allowing specification prototypes to be developed to test ideas at the earliest and cheapest opportunity. Specifications written in the technique may be subject to analysis methods to prove properties about the specifications, before implementation commences, thus saving on

E. Najm et al. (Eds.): FORTE 2006, LNCS 4229, pp. 307–322, 2006.

testing and maintenance time and providing a high confidence in the quality of the product to be developed. However these analysis methods are efficient only if they are supported by tools: for example, CPN-AMI [25], GreatSPN [9], PEP [13], CPNtool [22], automate the analysis process.

A problem with Petri nets is the explosion of the huge number of elements when described in their graphical form, for specification of complex systems. High-level Petri Nets [17] were developed to overcome this problem by introducing higher-level concepts, such as the use of complex structured data carried by tokens, and using algebraic expressions to annotate net elements. The use of high-level concepts within this Petri net framework is analogous to the use of those in high-level programming languages (as opposed to assembly languages). In the Petri nets community the term High-level net is generally used to refer to nets using such concepts.

Two of the early forms of high-level nets are Predicate-Transition Nets [12] and Coloured Petri Nets [16], first introduced in 1979 and further developed during the 1980s. Most of nowadays high-level nets build on these. They also use some of the notions developed for Algebraic Petri Nets [29], first introduced in the mid-1980s.

Furthermore, there are many different variants of Petri nets. Extensions of the technique, including time, stochastic features, capacities, and hierarchies as well as special Petri net types exist in the literature (see [2,8]...).

Standardisation of the technique has been seen as an opportunity to obtain a better organisation of the work in the Petri Net community. It has several issues:

- to enable the stakeholder — researchers, as well as engineers using Petri nets — to use the same terminology;
- to develop future extensions on a stable common basis, e.g., P/T nets or High-level nets;
- to provide a reference implementation that will facilitate the data exchange between different Petri nets tools through a common format.

The purpose of this paper is to present the PN standard, referenced under ISO/IEC 15909, as well as related work. First, the standard, which is organised in three different parts, is described. Then the current status of the work is given, followed by an implementation which is expected to prove very useful for the PN community.

## 2 The Structure of the Standard

The PN standard has been designed into three independent parts in order to enable flexibility of the standardisation process.

Part 1 [15] provides the mathematical definitions of High-level Petri Nets, called the semantic model, the graphical form of the technique, known as High-level Petri Net Graphs (HLPNGs), and its mapping to the semantic model. Part 1 also introduces some common notational conventions for HLPNGs.

Part 2 [18] [19] of this international standard defines a transfer format in order to support the exchange of High-level Petri Nets among different tools. This format is called the Petri Net Markup Language (PNML). Since there are many different versions of Petri nets in addition to High-level Petri Nets, this standard defines the core concepts of all Petri Net types along with an XML syntax, which can be used for exchanging any kind of Petri Net. Based on this PNML core model, part 2 also aims at defining the transfer syntax for the two versions of Petri Nets that are already defined in part 1 of this International Standard, Place/Transition Nets and High-level Petri Nets.

An addendum to Part 1 [20] of this standard introduces Symmetric Nets, formerly known as Well-Formed nets [6], as a subclass of High-level Petri Nets, which uses a restricted set of algebraic operators and allows for good analysis possibilities.

Part 3 is devoted to the standardisation of Petri nets extensions, including hierarchies, time and stochastic features. These extensions will be built upon extensions of the core model. They require a stable version of the core model to be available. This is not the current situation at this stage. Hence, only parts 1 and 2 are presented below.

The standardisation process is quite long and relatively complex. A standard must be built in order to be stable enough to be used by the people involved. It is developed within a schedule which should not exceed three years and is subject to revision every five years. More information on the rules can be found at [10].

Part 1 obtained the status of International Standard in december 2004. The addendum has been proposed by France and has currently the level of Working Draft (stage 20.60 in the ISO nomenclature).

At this level, the possibility is offered to the community to contribute. When the standard has reached the step forward, the Committee Draft level, it gains restricted aceess, with rights reserved to ISO experts only.

Part 2 has today the same status as the addendum.

Work on part 3 has not started yet, as it requires a stable version of the PNML core model to be available. As a consequence, the work on this part will start as soon as PNML is standardised.

### 2.1 Part 1

The first part of ISO/IEC 15909 was published as an International Standard (IS) in december 2004. It provides a comprehensive documentation of the terminology, the semantical model and the graphical notations for High-level Petri nets. It also describes different conformance levels. Finally, a tutorial example given in annex illustrates the different concepts in the standard.

A *glossary* introduces the different terms to be used in the Petri net context. They thus have a precise meaning, which is explained in natural language in the glossary and further detailled later using mathematical notations. The document is thus self-contained and avoids any ambiguity.

The *semantic model for High-level Petri nets* is defined, using precise mathematical notations. All basic elements required to work with High-level Petri nets

are thus introduced: *high-level Petri net*, *marking*, *enabling* of transition modes, and *transition rule*.

These mathematically defined concepts are then reintroduced using natural language and explanations, and related to the *graphical notations* which are more commonly used in practice. Hence, the graphics representing the nets are defined.

This graphical presentation is further formalised as a *High-level Petri Net Graph*. It also has a semantics. It can be viewed as a graph oriented perspective for the high-level Petri net semantic model.

An important issue in standards design is the conformance level. Indeed, other work or tools can be compliant with the standard as a whole, or just part of it. This latter case may be sufficient for some particular purposes. Different conformance levels are thus defined, both for Petri nets and High-level Petri nets, depending on whether the graphical notation is taken into account.

Extensive mathematical notations are defined as normative in an annex. Another normative annex defines net classes. Up to now it only comprises *Place/Transitions nets* (i.e. Petri nets). Another class definition for *Symmetric nets* (formerly known as Well-Formed nets) is currently in the process of being an addendum to part 1 of the standard.

### 2.2 Part 2

The objective of part 2 is to define an interchange format for Petri nets called PNML (Petri Net Markup Language) [3]. This interchange format relies on XML technology.

However, designing an interchange format in the context of this standard is a difficult task since part 3 will introduce more Petri net types. It is obvious that an exchange format only suitable for the Petri net types defined in part 1 is not appropriate. This problem was already outlined in a preliminary study in 2000 that was classifying tools according to the type of supported Petri nets [1].

Moreover, tools usually introduce small variations in Petri nets and create their own "dialect". These variations are mainly due to syntactical aspects, to some graphical facilities or the way "actions" are added to the specifications. Actions are a way to provide help to the system designer, for example by making available instructions to ease animation of the specification, or add breakpoints.

So, to cope with all these goals, PNML must be able to:

1. allow to introduce smoothly new information associated with new Petri nets types or, by restriction, allow to hide some information from an inherited Petri net class.
2. support data aside of the standard, to let tools supporting non-standard extensions of another tool be able to handle it.

We provide hereafter some details about these two points. The next section will provide information about the way we handle them appropriately in the standard by using model engineering techniques implemented using EMF [11] technology from Eclipse.

*Handling a hierarchy of Petri net types.* Our first problem is related to the adjunction of new Petri net types in part 3 of the standard. Let us consider the small hierarchy expressed in figure 1. *P/T nets* are the root class since they only define the basics of Petri nets. Then, they can be extended to *Symmetric nets* proposed to be an addendum to part 1 [20]. Since Symmetric nets are a restriction on the color functions and types allowed in a *High-level net*, there is another trivial relation to them.

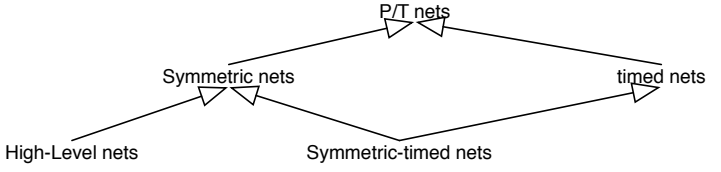


**Fig. 1.** Example of Petri net types hierarchy

Let us now consider another set of features in Petri nets: time management. So, *timed nets* can be derived from P/T nets by adding the time information to transitions as in [2]. We can also consider that *Symmetric-timed nets* inherit from both timed nets and symmetric nets.

The interchange standard must be flexible enough so as to allow any conversion from one of these representations to any other one without loosing information when the tools do handle them. It is crucial that the standard is able to handle a hierarchy of Petri net types.

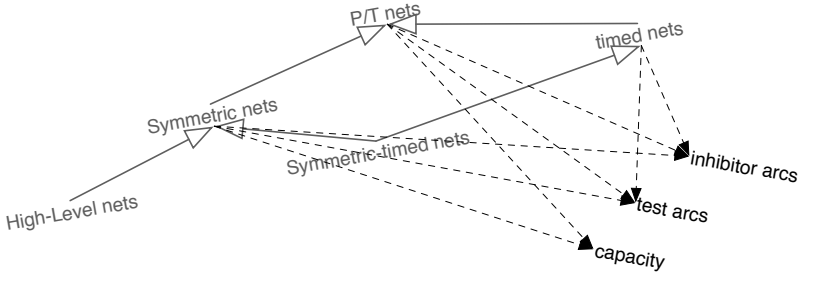


**Fig. 2.** Connections of the Petri net hierarchy to "local variations"

*Handling small variations in Petri net types.* Our second problem is to deal with local variations within a Petri net type such as inhibitor arcs, capacity in places or any other tool specific information (such as graphical specificities). This is illustrated in figure 2. We consider there variations such as inhibitor and test arcs, as well as capacity places. Such variations can be operated for several types of Petri nets. In our figure, we consider they are all relevant for Symmetric nets and for P/T nets. Only inhibitor and test arcs are also considered for timed nets.

Once again, the standard must be able to cope with such variations at various levels in the Petri net types hierarchy. It is important to normalize as many variations as possible to have them compatible all over the Petri net hierarchy.

## 3 Current Implementation of Part 2

The second part of the standard defines a universal transfer format (PNML) for exchanging Petri net models among Petri net tools. Hence, its primary purpose is to enable *interoperability*.

In this section, we first of all highlight how PNML design is being carried out through the specification work on the standard. Then, we introduce the incentives for the first implementation of a translation software framework to back the standard, using model engineering techniques.

### 3.1 PNML Design

The adopted methodology to design PNML is structured in two main steps:

1. the *abstract syntax* definition through Petri net types definition with metamodels;
2. the *concrete syntax* definition by mapping the abstract syntax onto PNML schema.

During the first step, main Petri net types are defined using *metamodeling techniques*. It means that we describe the concepts and rules structuring these types and their meaning, at a high level of abstraction, independently from any technological choice for their future implementation. Metamodeling is always purpose- or business-oriented. It is an activity during which experts of a particular domain define the precise semantics of the specific concerns they are interested in. For example, business process modelers might design a workflow metamodel for a supply chain, the purpose of which is to discover where synergies could be gained.

Following our motivations stated in section 2.2, it is important, using such techniques, to reach a sufficient level of abstraction in PNML design. Indeed, we should be able, when further developing the standard (Part 3 and maintenance updates of all parts), to easily refine and extend the primary specifications to define new types or variants of Petri nets. Therefore, it would be useless to fall at first in a too low level of specifications, from which no valuable abstraction could be made to improve the standard.

Three main Petri net types are defined. They are described using the Unified Modeling Language (UML) class diagrams. They are:

1. The *Core model*. It is the most fundamental one, depicted by fig. 3. Core concepts of Petri nets can be found in this basic type: nodes, connectors, basic labels (e.g., names) and graphical information associated with these objects. It provides the foundation for further definition of new Petri net types.

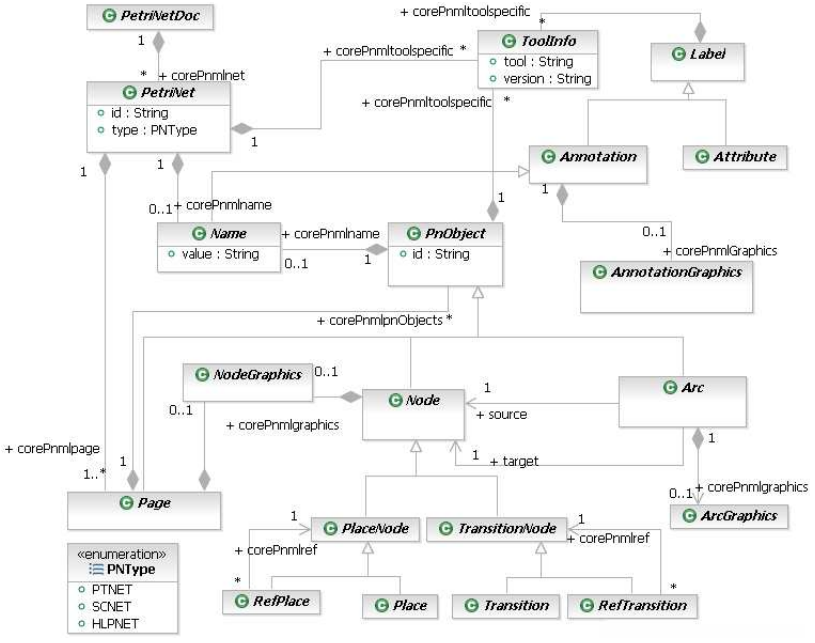


**Fig. 3.** PNML Core model

2. *P/T Systems metamodel*. It essentially defines new labels for this type of Petri nets and relies on the Core model for the central concepts. It is thus built upon extensions of the Core model.
3. *High-level Petri Nets metamodel*. Its design is in progress. New labels and high-level functions are being defined, while relying on the concepts provided by the Core model and P/T Systems type. As a consequence, it is also built on extensions of P/T Systems.

An important aspect of this first stage is to state semantic constraints on the metamodels. This may be achieved using the Object Constraint Language (OCL) [24]. For instance, in P/T Systems, two connected nodes must not be of the same kind (i.e., no place-place or transition-transition arc is permitted).

During the second step, PNML schema is defined to match the specifications carried out by the metamodels. Technical details set apart, each relevant element of the metamodels is mapped onto a PNML tag.

PNML schema technology is currently *RELAX NG*-based, which is "being developed as an International Standard ISO/IEC 19757-2" [7]. RELAX-NG is XML schema-like, more flexible and maintenance-friendly for the standard designers than XML schema. However, coming to High-level Petri Nets type, using this technology to directly define a notation to express high-level labels and functions does not seem appropriate. Consequently, MathML [31] is being investigated since it seems to offer the complete set of annotations that could be used to define high-level labels and functions. We are taking a particular care in

extracting the most accurate subset of MathML features since it allows for great flexibility and powerful expressivity.

### 3.2 The First Impact: PNML Framework

From the motivations reported in section 2.2 and emphasized through the design methodology of PNML described in section 3.1, it is clear that the *extensibility* issue for the standard definition is of central importance.

**Motivations.** In addition, there are three other important issues we should cope with that drive the implementation of a translation software framework to support the standard.

*Semantic constraints.* The Petri Net Markup Language should carry the syntax of Petri net types specified by the standard. However, capturing the semantics is also an important issue. Moreover, how to ensure that it is enforced? Semantic constraints are expressed in the metamodels by means of OCL [24] statements. But it is important to note that OCL is a specification language, not a programming one. Therefore, it is not meant to be directly executable. It takes its full meaning when associated with UML annotations.

*Compatibility.* A second important issue pointed out by PNML specifications is the compatibility among Petri net types and their variants (cf. discussion in section 2.2 related to Petri net types hierarchy). Since all variants of the main types may not be specified by the standard, how to continuously ensure the interoperability and thus the exchange of Petri net models ?

Another related issue is the compatibility between PNML successive versions. For example, at least a top-level *Page* is mandatory in the current version under development, unlike the previous one.

*Automation and integration in Petri nets tools.* Since PNML is XML-based, it is error-prone to manual editing. Therefore, it obviously needs an application to perform this task.

To ensure that 1) all issues we pointed out are equally dealt with, and 2) to favor an up-to-date compatibility with the standard along with 3) an easy integration of its implementation into Petri net tools, we have developed a model-based translation framework to back the standard.

In the following, we describe this framework, called *PNML Framework*, and its use.

## 4 PNML Framework

To make the standard applicable and provide a reference implementation to Petri net tools developers, we have developed PNML Framework. Its first release was published in March 2006 [21].

In this section, we first present this tool and the benefits it offers to tools developers. Then we describe its features and give an overview of its use. Eventually we conclude by sketching further work. Open issues are discussed in section 5.

### 4.1 Goals of PNML Framework

The primary aim is to provide efficient and standard compliant import and export features of PNML models for Petri net tools. Figure 4 illustrates how PNML Framework could make the standard interoperability goal achievable. It shows two tools, $A$ and $B$, exchanging a Petri net model via the standard transfer format, using the framework. More details about the operations involved are given in sections 4.2 and 4.3. PNML Framework is designed using model engineering techniques and, more precisely, EMF (Eclipse Modeling Framework) [5] technology.

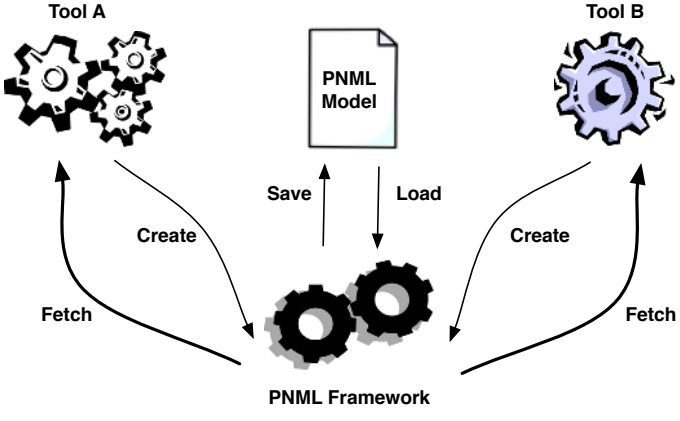


**Fig. 4.** Interoperability using PNML Framework

PNML Framework is a generated set of comprehensive and easy to use tailored API to import and export Petri net models designed according to the standard specifications. It is intended to be used as a library, therefore it can be easily integrated into Petri net tools. Tools developers are considered as the primary users of PNML Framework.

Thanks to this framework, tools developers would rather focus on their applications core development instead of coping with how to stay up-to-date and compliant with the standard. Furthermore, they would not have to deal with ensuring continuous compatibility with other tools and many Petri net types and variants.

Moreover, PNML Framework's flexibility enables them to export and import appropriate elements of Petri net models, according to their needs. For example, after having loaded (this term is explained in the following sections) a high-level net into the framework, one can only fetch the P/T net associated structure.

*How is it generated?* The API is generated from the standard metamodels using EMF's tools.

First, metamodels from the standard are implemented in EMF's ecore modeling language [5]. In its modeling approach, EMF can be seen as an optimized

implementation of OMG's Essential Meta Object Facility (EMOF) [23] specification but with some differences, due to the experience gained from at least five years of development and wide use.

After having designed the metamodels using ecore, code is then generated so as to enable manipulation of model instances of these metamodels. To meet PNML specific requirements we have extended the Java Emitter Templates [27] code generation tool integrated in EMF. Consequently, the generated code is completely tailored to PNML. PNML Framework is thus extensible to include any Petri net type since its implementation is model-driven and follows the requirements expressed in the previous sections.

### 4.2 Features of PNML Framework

The design of PNML Framework is model-driven. It is supported by EMF, which is a mature model-driven application development framework. It implements Petri net types metamodels defined in the standard. It handles PNML models which are instances of these metamodels. From the framework's point of view the representation of a PNML model is twofold:

- it is a Petri net model which is an instance of a Petri net type metamodel; in that case it is handled in memory;
- it is a Petri net model written in PNML (XML-based) syntax in a PNML document. In that case it is an instance of PNML schema. A PNML document can contain one or more PNML models. PNML schema is the RELAX NG-based XML schema which is mapped onto Petri net types metamodels. It describes the concrete syntax of PNML models.

As a benefit of using model engineering techniques, PNML Framework offers two well-defined principal features in the context of PNML models:

- **export**: PNML models are *created* as instances of Petri net types metamodels and *saved* in PNML syntax;
- **import**: PNML models are *loaded* from PNML documents and created as instances of Petri net types metamodels in the framework. Their elements are *fetched* by the framework's user (tool developer).

These features are offered through the API which is structured in four sections:

1. **Create**. This section entitles tools developers to translate user-defined Petri net models represented in their proprietary format into PNML models as instances of Petri net types metamodels.
2. **Save**. It is used to save created PNML models into PNML syntax in PNML documents. Tools developers are offered a single method in this section to trigger that operation.
3. **Load**. PNML models are read by parsing PNML documents and loaded into the framework by using the *Create* section. Here again, a single method is provided in this section to perform the transfer.
4. **Fetch**. It is used to retrieve elements of PNML models that have been loaded.

In addition to these features, we have implemented rules to enforce semantic constraints on PNML metamodels that are expressed by means of OCL [24] statements in the standard. Therefore, users should not have to cope with how to integrate these constraints since the framework natively implements them.

### 4.3 Using PNML Framework

Figure 5 describes typical interactions between a tool developer's application core program and PNML Framework. In this figure, the four sections of the API label the interactions. *My model* represents a Petri net model in a proprietary format. *My Program* is the tool developer's core program which drives the model translation from the proprietary format into PNML syntax. It uses PNML Framework as a library to perform this task.

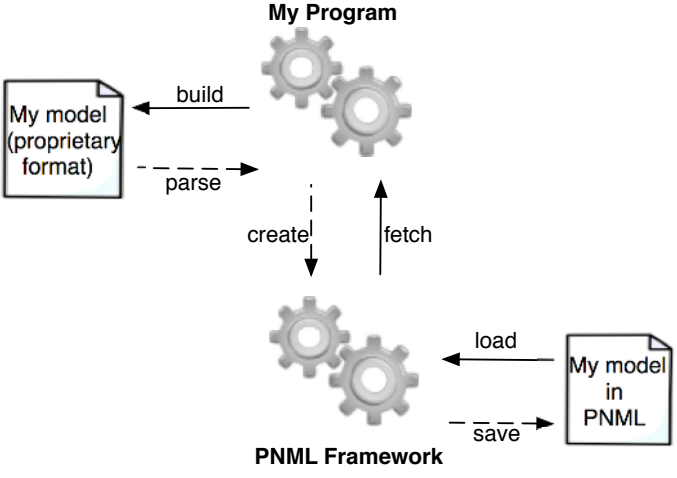


**Fig. 5.** Overview of tools developers' use of PNML Framework

To export a Petri net model represented in a proprietary format or handled in any application into PNML, a tool developer may first be entitled to *parse* it. Then, using the predefined *create* API, the corresponding PNML model can be created in the framework. Eventually the model will be *saved* in PNML syntax.

To import a Petri net model represented in PNML syntax (*My model in PNML* in fig. 5), the framework first *loads* that model. Then, using the predefined *fetch* API, tools developers can retrieve its elements and *build* the corresponding model represented in their proprietary formats or perform another task.

Let us show an example of a P/T model exchanged between a proprietary format and PNML, using PNML Framework. In this example, we focus on the export feature from the proprietary format into PNML. That format, called CAMI, is used in our CASE environment, CPN-AMI [25]. We have developed an application, named PNML Converter, which uses PNML Framework *create* and *save* API to perform the export.

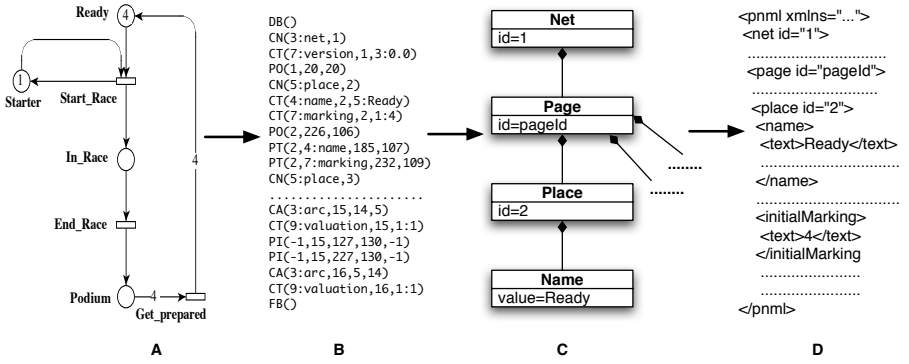


**Fig. 6.** An example of model transformation from CAMI to PNML

In fig. 6, P/T model $A$ is the graphical representation of a Petri net model created in CPN-AMI environment. The graphical representation is transformed into model $B$, the syntax of which is CAMI. Then using PNML Framework's *create* API, it is transformed into a model instance (called $C$ in the figure) of the standard-defined P/T Systems metamodel [18]. Finally, using the *save* API, it is transformed into PNML syntax, as model $D$. Subsequently, there were actually three model transformations. The first two ones are under the responsibility of the tool designer, the role of whom we took in this example. PNML Framework is in charge of the last one.

PNML Framework is packaged as a Java library. It provides command-line invocation features to ease its integration in tools. For instance, once *My Program* packaged with the framework as a complete translation application, that application becomes a service for the considered tool. It will then be invoked by providing a simple API to develop driver encapsulating users' invocations caught through tools interfaces. We have implemented such an approach for CPN-AMI [25] and it was successful.

### 4.4 Current Work

Presently, P/T Systems are supported by the framework. The core model is of course implemented. But since we do not consider it is a concrete Petri net type suitable for exchange among Petri nets experts, we do not offer the possibility to export and import it explicitly by the framework. Indeed, as explained before, the core model is intended to set a strong basis for the definitions of concrete Petri net types.

In further developments of this framework, we are implementing Symmetric Nets. Symmetric Nets are annotated with higher-level labels and functions. Those labels are being defined using MathML, as stated in section 3.1. In the next section we discuss related issues to this work.

We are also planning to develop an advanced version of PNML Framework, which will generate specific APIs for local variations on Petri net types, not

specified by the standard. This will help tools developers with very specific needs to exchange their models, provided that they share the newly generated APIs. The normal version of the framework which is fully standard compliant will simply discard these particular local variations. The use of this advanced version requires deep knowledge of metamodeling and code generation techniques using EMF's powerful features.

## 5 Open Issues for Future Work

As mentionned in the previous section, high-level labels and functions are defined using MathML. However, since MathML is very expressive, there are often different equivalent ways to define the same function. Consequently, it is prone to break the interoperability and compatibility objectives through the non-unification of the semantics, if MathML is used "as is". Moreover, in exchanging automatically processed PNML higher-level models, we cannot expect a meaningful interoperability to take place. Let us recall that metamodels should always carry as much as possible a precise semantics for a specific purpose. To make high-level labels and functions use both unambiguous and fully understandable, we should define the metamodel that carries their semantics for higher-levels of Petri nets (Symmetric Nets, High-level Petri Nets). This is of utmost importance since part 3 of the standard will introduce new types of Petri nets.

Our approach to tackle this issue relies on an unambiguous schema of a subset of predefined labels and functions expressed in MathML to be defined in the standard. This schema would be mapped as a concrete syntax to the corresponding high-level annotations metamodel for higher-level Petri nets. This schema corresponds to a normalised way to define abstract syntax trees for complex labels (independent from any technology implementing the syntax). This approach is consistent with the methodology adopted for the definition of the second part of the standard, described in section 3.1. To ensure interoperability and compliance with the standard, tools developers must enforce the use of this predefined subset. Therefore we propose to ease the application of this requirement by integrating its implementation in a future version of PNML Framework.

It is of interest to experiment first this strategy to Symmetric Nets: this type of Petri nets only allows for a restricted set of algebraic operators and no user-defined function.

High-level Petri nets allow for a larger set of algebraic operators than Symmetric Nets, but the user can also define his/her own functions. This last point may lead to ambiguous representation of these functions if no normalised action (such as strict structuration of MathML expressions) is considered. This is also a challenge, when we consider that part 3 of the standard will introduce new types of Petri nets.

## 6 Conclusion

A survey of the standardisation work on Petri nets, known as ISO/IEC-15909, is presented in this paper. The standard is structured into three parts.

Part 1 is now an International Standard. Part 2 is currently under development. It provides the abstract definitions of significant Petri net types and their concrete syntax. This syntax is intended to be a universal transfer format to enable interoperability among Petri net tools. A wide adoption of the standard among Petri net experts can thus be reached. It is called Petri Net Markup Language (PNML). Part 3 will rely on definitions carried out by part 2. It will define new types and variants of Petri nets. When a stable version of part 2 is reached, the work on part 3 will start.

We are also experimenting an implementation of part 2 in PNML Framework. PNML Framework primary purpose is to make the standard applicable. Therefore it puts the interoperability goal into action. It offers Petri nets tools a flexible way to remain up-to-date and comply with the standard while dealing with extensibility, compatibility and semantic issues. To cope with such issues, we are using model engineering techniques to sustain PNML Framework development. In [14], we set the rationale for this approach.

The first release of PNML Framework was published in March 2006 [21]. It is implemented in Java, to achieve the cross-platform objective expressed in PNML earliest requirements. We provided in this release a tool developer's guide and a tutorial. We also provided an application example of conversion using GraphViz [30] *dot* format, to ease the full understanding of PNML Framework's capabilities:

- efficient model-driven import and export tool for PNML models,
- standalone execution (outside Eclipse);
- easy integration in Petri net tools.

We are currently enhancing PNML Framework with a new type defined in the standard: Symmetric Nets. This will assess the consistency of our approach in PNML Framework design. Symmetric Nets are a first step towards the support of High-level Petri Nets in the standard. We also take into account the Petri net community feedback.

It is of interest to set up a prototype implementation project from part 2 of the standard in PNML Framework. It contributes to establish a meaningful assessment of the standard implementation and use in the context of tools design.

## Acknowledgements

We would like to thank ISO/IEC 15909 editors, especially Jonathan Billington and Ekkart Kindler, for the insightful discussions that helped us enhancing this paper.

## References

1. R. Bastide, D. Buchs, M. Buffo, and F. Kordon adn O. Sy. characteristics of currently used petri nets. Technical report, Univ. P. & M. Curie, available at `http://www-src.lip6.fr/homepages/Fabrice.Kordon/PN_STD_WWW/Qresult.html`, 2000.
2. Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.

3. J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, technology and tools. In *Proc. 24th Int. Conf. Application and Theory of Petri Nets (ICATPN'2003), Eindhoven, The Netherlands, June 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–505. Springer, 2003.
4. Brauer, W., Reisig, W., and Rozenberg, G., editors. *Petri Nets: Central Models and Their Properties.*, volume 254. Springer-Verlag Lecture Notes in Computer Science: Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986, 1987.
5. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T.J. Grose. *Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley Professional, August 2003.
6. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and their symbolic reachability graph. In G. Rozenberg and K. Jensen, editors, *LNCS : High Level Petri Nets. Theory and Application.* Springer Verlag, June 1991.
7. J. Clark. *RELAX NG Home Page*. OASIS, `http://www.relaxng.org/`, 2003.
8. M. Diaz. *Vérification et mise en oeuvre des réseaux de Petri*. Hermes Sciences - Lavoisier, 2003.
9. GreatSPN: GRaphical Editor, Analyzer for Timed, and Stochastic Petri Nets. url: `http://www.di.unito.it/~greatspn/`.
10. International Organization for Standardization. *International harmonized stage codes*. ISO, http://www.iso.org/iso/en/widepages/stagetable.html#95.
11. Eclipse Foundation. *Eclipse Modeling Framework*. `http://www.eclipse.org/emf/`.
12. H. J. Genrich. Predicate/transition nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Lecture Notes in Computer Science: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986*, volume 254, pages 207–247. Springer-Verlag, 1987. NewsletterInfo: 27.
13. Parallel Systems Group. *Programming Environment based on Petri Nets*. University of Oldenburg, `http://theoretica.informatik.uni-oldenburg.de/~pep/`.
14. L. Hillah, F.Kordon, L. Petrucci, and N. Trèves. Model engineering on petri nets for iso/iec 15909-2: Api framework for petri net types metamodels. *Petri Net Newsletter*, (69):22–40, October 2005.
15. ISO/IEC. Software and Systems Engineering - High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, International Standard ISO/IEC 15909, December 2004.
16. K. Jensen. Coloured petri nets - basic concepts, analysis methods and practical use, vol. 3: Practical use. *EATCS Monographs on Theoretical Computer Science*, 1997.
17. Jensen, K. and Rozenberg, G., editors. *High-Level Petri Nets.* Berlin, Germany: Springer-Verlag, 1991. NewsletterInfo: 39.
18. E. Kindler. Software and Systems Engineering - High-level Petri Nets. Part2: Transfert Format. Working Draft for the International Standard ISO/IEC 15909 Part 2 - Version 0.9.0, June 2005.
19. E. Kindler. The petri net markup language and iso/iec 15909-2: Concepts, status, and future directions. In *Entwurf komplexer Automatisierungssysteme*, To appear.
20. F. Kordon and L. Petrucci. Proposal for an addendum to ISO/IEC 15909-1, document reference MAL-12. NWI For the Malaga Meeting, November 2004.
21. Modeling and Verification Department. *PNML Framework*. LIP6, `http://www.lip6.fr/pnml`.

22. University of Aarhus. *Computer Tool for Coloured Petri Nets - CPNTool.* http://wiki.daimi.au.dk/cpntools/cpntools.wiki.
23. OMG. *MetaObjectFacility 2.0 Core Specification, document no:omg/2003-10-04.* OMG, October 2003.
24. OMG. *OCL 2.0 Specification - Version 2.0 ptc/2005-06-06.* OMG, June 2005.
25. The CPN-AMI Home page. url : http://www.lip6.fr/cpn-ami.
26. J. Peterson. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.
27. Remko Popma. *Introduction to JET.* Azzurri Ltd., http://eclipse.org/emf/docs.php?doc=tutorials/jet1/jet_tutorial1.html, 2005.
28. W. Reisig. *Petri Nets.*, volume 4. Springer-Verlag EATCS Monographs on Theoretical Computer Science, original edition, 1985. NewsletterInfo: 19 translation of the German: "W. Reisig, Petrinetze. (1982)".
29. W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991. NewsletterInfo: 38,39.
30. AT&T Research. *GraphViz.* http://www.graphviz.org/.
31. W3C. *MathML 2.0, W3C Math Home.* W3C, http://www.w3.org/Math/.