

# Building an API for ISO/IEC 15909, based on model engineering techniques

Lom Hillah, Fabrice Kordon (LIP6, Univ. P. & M. Curie),  
fabrice.kordon@lip6.fr  
Laure Petrucci (LIPN, Univ. Paris-XIII),  
laure.petrucci@lipn.univ-paris13.fr  
Nicolas Trèves (Cedric-CNAM)  
treves@cnam.fr

May 22, 2005

## Abstract

Part 2 of ISO/IEC 15909 standard aims at defining a tool interchange format for Petri nets, PNML. It is XML based, and shall support different classes of Petri nets, be extensible to allow for future classes. In this paper, we propose a software framework to derive APIs from Petri net classes metamodels. This is motivated by a concern for compliance with the standard, enhancing compatibility between tools. The production chain envisioned is based on model-engineering techniques. We detail the different steps, the input metamodels and the outputs.

## 1 Introduction

The Petri Net standard (ISO/IEC 15909) contains 3 parts. Part 1 [6] provides a reference definition for Petri nets. It is now in the final stage of the process of becoming an international standard.

Part 2 deals with the definition of PNML, a tool interchange format dedicated to Petri Nets. PNML is based on XML to support several classes of Petri Nets (so far, three have been identified: Place/Transition nets, Well-Formed Petri nets and High-Level nets). Up to now, part 2 is a draft standard, basically based on a proposal discussed during the International Petri Net Conference in 2003 [4].

In parallel to part 2, part 3 of the standard is being started. It focuses on modularity and Petri net extensions (e.g. time).

Part 2 is therefore today the part of the standard where the most active work is expected. PNML relies on:

- the PNML Core Model, a Petri Net metamodel (represented as UML class diagrams) based on a PNTD (Petri Net Type Definition). This core model consists of a graph structure compatible with any kind of Petri net.
- a Petri Net Type Definition Interface (PNTDI), which is the mechanism to define the format for any type of Petri Net.

The PNML syntax has been defined in terms of RELAX NG grammar, to enable the translation of a given Petri net into an XML representation.

To be useful, an implementation of these concepts is required with a particular care w.r.t. two types of versatility induced by:

- the variety of Petri net classes that will be considered in part 3 of the standard.
- tools that are usually based on a Petri net class on which specific information may be plugged to enable the use of tool-specific features (e.g. reference to source code).

This paper intends to propose a *software framework* to support Petri net tools interoperability. This framework is based on the definition of a standard API supporting PNTDI and built using the MDA transformation approach [9].

The paper is structured as follows. Section 2 summarises PNML main goals and sets our ideas within the context. Then, section 3 recalls the principles of model engineering and model transformation techniques. It sets our work on PNML w.r.t. the MDA approach. This is further detailed in section 4 with both the requirements and the functioning of the production chain to produce APIs. Its inputs are the Petri Net Core Metamodel and extensions which are explicated in section 5. Finally, the code generation process is described in section 6.

## 2 Context

Petri Net Markup Language (PNML) is under standardisation process as ISO/IEC 15909 part 2. It aims to be an XML-based standard interchange format for Petri nets in order to provide interoperability between various Petri net tools. Since ISO/IEC 15909 part 3 will introduce more Petri net types, PNML should also allow for extensibility and compatibility.

So, PNML should have the following capabilities:

**Compatibility** This PNML standard shall allow for interoperability, and be compatible with all classes of Petri nets.

**Extensibility** It shall be versatile and extensible, so as to include other variants of Petri nets developed in the future.

**Applicability** It must allow engineers to develop their Petri nets tools interfaces using the main high-level programming languages, e.g. Java, C++, Ada, C.

*Compatibility* is enforced by using a de facto standard: XML.

To enforce *extensibility*, we must propose to use a core Metamodel corresponding to P/T nets. Then, we should provide a methodological approach to define extensions of the core Metamodel to represent higher level types of Petri nets (such as inhibitor arcs, capacity places, low-level priorities, etc.). Such an approach should take place within the context of MDA [9].

To make the standard *applicable*, we propose a software framework, designed using the MDA approach. It provides tools designers with standard APIs to manipulate Petri net models. These APIs are automatically generated from the corresponding metamodel.

MOF and XMI are appropriate candidates to follow a MDA approach. Hence we propose to base our design approach on such standards. We use XMI to represent PNML and MOF to specify both the Core metamodel and extended ones.

Designing a program to load/save Petri net files according to the standard interchange format is an error-prone task and may lead to problems of compliance with the standard. To avoid such problems, we propose to follow the production scheme pictured in figure 1.

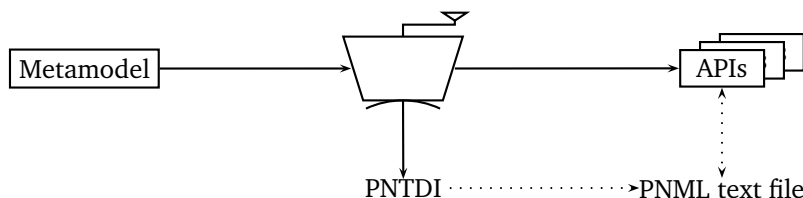


Figure 1: Production process

The process starts from a metamodel comprising two parts: the *Petri Net Core Metamodel* (PNCM) and a *Petri Net Type Definition* (PNTD). The PNCM describes how a basic Place/Transition net is represented, according to the standard. This part is the core of the interchange format and should be fixed whatever the Petri net type or tool. Conversely, the PNTD contains additional information that enhances the selected Petri net class. It can then be seen as an add-on to the norm intended for a particular type of Petri net (e.g. a net with inhibitor arcs).

Then, the metamodel is processed in order to produce both a *Petri Net Type Definition Interface* (PNTDI) which captures all the information in the metamodel, and APIs in main programming languages. Thus the code generation is an automatic process.

The APIs and PNTDI allow a tool designer to directly include the code for loading/saving PNML models as text files in his/her own tool. Moreover the tool designer could enhance the Petri net description at several levels in order to meet his/her needs. Either the changes affect the model semantics, and then *major extensions* should occur at the metamodel level, or *minor extensions* (such as tool specific tags like a particular graphical feature) which do not modify the model semantics are inserted at the API level.

This production chain has several pros and cons. It benefits from model engineering techniques, facilitates the development process. A drawback is that produced PNML files are verbose. However, considering both the introduction of Petri nets in UML 2.0 and the increasing use of model engineering techniques for model transformation in a MDA-based development process, we think it is of interest to have such a production approach. PNML is more than an interchange format between tools. It is also a strong basis to associate Petri nets with classical tools and techniques issued from model engineering to achieve model transformation as presented in [5].

### 3 Model engineering and model transformation

In 2001, the OMG introduced MDA [9] that promotes system development at a model level. MDA introduces several successive models: PIM (Platform Independent Model) and then PSM (Platform Specific Model). To cope with transformation from one model to another, model engineering techniques propose a set of solutions that are being studied and normalised by the OMG.

### 3.1 Overview of model transformation

Model engineering techniques are concerned with models transformations. These techniques require the definition of metamodels and sets of rules to perform a transformation from a source metamodel to a target metamodel. Such rules define the *metamodel mappings*. Then, any model compliant with the source metamodel (i.e. formalism) can be processed by a given set of rules, thus rigorously defining a transformation towards the target metamodel.

We use an MDA-based approach to tackle our issues. MDA philosophy is aimed at portability, interoperability and reusability through architectural separation of concerns.

Generally speaking, given a source metamodel and a target metamodel, transformation rules could be input to an engine, so as to perform transformation on a given source model in order to obtain the target model.

We first intend to specify a core metamodel for Petri nets, the *Petri Net Core Metamodel* (PNCM). This core metamodel is based on P/T nets, as defined by ISO/IEC 15909 Part 1 standard. Its extensibility should allow for refinements by introducing features related to other classes of Petri net, among which we will consider High-Level Petri Nets and Well-Formed Petri Nets, since they are part of the ISO standardisation process.

From a specified Petri net metamodel, a PNML data representation could be derived, hence defining a *Petri Net Type Definition* (PNTD), according to the PNTDI<sup>1</sup>. This is the RELAX NG based grammar, used for defining the format of a particular version of a Petri net. As for classical XML document elements that refer to a XML schema, a PNML Petri net model representation refers to a Petri Net Type Definition (i.e. a particular class of Petri nets) that enables the use of its model elements. For instance, a P/T net model derives from the PNCM and its PNTD is under standardisation process.

The next step is to build an API into a targeted language for the user's application needs. To meet the MDA approach guidelines, we could specify an abstract API and apply transformations techniques by setting up rules as input to a metamodel transformation engine. The abstract API could then be derived into a targeted language, such as Ada, C++ or Java, thus enabling the user's tool to perform engineering design operations on models: creation, loading, manipulation and storage. This approach will prevent a user from writing PNML models by his/her own means, and thus avoid inconsistencies.

Industrial practitioners acknowledge the advantages of MDA-based development techniques for designing complex systems. Separation between domain and platform specific issues, namely PIM and PSM, allow a more precise approach to the viewpoints of the system under design. The framework provided by the OMG, MOF [8], includes two viewpoints :

1. *the modelling viewpoint* : it deals with the meta levels and hence provides model information for a particular field. For example, in this paper, we specify a metamodel for Petri Nets, according to the ISO/IEC 15909 standard. Then we provide specific extensions to this metamodel to allow for its versatility, so as to define particular Petri Net classes, such as High-Level Petri Nets or Well-Formed Petri Nets.
2. *the data viewpoint* : the implementation level, from which APIs for different programming languages can be developed to meet specific requirements for the engineers' tools.

Hence, MOF aims at extensibility by providing a framework for representing any kind of metadata. The classical four layer architecture describing MOF concept is of interest. Here, we consider the two inner layers, metamodel layer (M2) and model layer (M1).

---

<sup>1</sup>Petri Net Type Definition Interface.

### 3.2 Metamodel specification viewpoints

Experiences such as the metamodeling of UML revealed that there are three main views, each corresponding to orthogonal goals (figure 2):

1. *abstraction*: it emphasises reusability and any type of extensibility for the specified metamodel. However, due to a high level of abstraction, it is difficult to express precise semantics in the metamodel. So, implementation is heavy and strongly reduces applicability.

This view has been experimented with UML and lead to dramatic complexity when enabling the respect of constraints by means of OCL assertions. This approach is very difficult to maintain.

2. *exchange*: its purpose is to allow metamodel exchange for technical discussion between system designers.
3. *productivity*: an emphasis is put on structuring the metamodel in order to set constraints without additional OCL assertions. Then, it is possible to constrain extensibility to appropriate directions (such as avoiding to connect two places together). This view guarantees a certain level of compliance with the core metamodel. A customisation process is easier to define.

APIs generated with such an approach are generally more accurate in terms of usability by programmers.

As shown on figure 2 we mainly target the productivity viewpoint, while remaining open to reusability and extensibility for the PNCM.

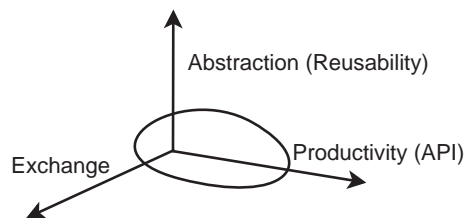


Figure 2: Metamodeling viewpoints

In our opinion, the metamodel sketched in the current draft version of ISO/IEC-15909 part 2 puts too much emphasis on the first axis. Since: 1) extensibility should be constrained in order to preserve the Petri net semantics, 2) APIs should be usable, we propose to rewrite the metamodel putting emphasis on the third axis. An updated version of the metamodel is presented in section 5.

## 4 Methodology

This section explains how we will derive APIs from the metamodel, using model engineering-based techniques.

## 4.1 Building blocks

To set a reference context to this proposal, we provide hereafter normative references and some definitions.

**Normative References** We acknowledge the following references:

- ISO/IEC 15909-1 Software and system engineering - High-level Petri Nets - Concepts, Definition and Graphical Notation,
- ISO/IEC 15909-2 High Level Petri Nets - Transfer Format,
- ISO/IEC JTC1/SC7 N3183 - Proposal for an Addendum to ISO/IEC 15909-1,
- UML,
- XML and XML Schema,
- RELAX NG.

**Terms and Definitions** All definitions from ISO/IEC 15909, Part 1 are adopted. We also consider definitions from part 2 and the *Proposal for an Addendum*. We introduce the following for model engineering purposes:

- PNCM : Petri Net Core Metamodel. It is based on PNML Core Model and Place/Transition Type Model, as specified in [7].

## 4.2 Working requirements

The main goal of this paper is to propose a software framework to support Petri nets tools interoperability.

Using engineering concepts, based on OMG MDA standards and MOF specification, we will derive an extensible multi-language API from Petri nets metamodels, providing PNML description of these, with respect to ISO/IEC 15909 Parts 1 and 2 standard description of Petri nets.

The PNML core metamodel remains unchanged. Systems designers may use this core metamodel to generate an API to manipulate their models. They may define their own PNML metamodel, taking into account a specific type of Petri net with new features, such as High-Level Petri Nets or Well-Formed Petri Nets <sup>2</sup>.

The PNML metamodel is described by a *Petri Net Type Definition* (PNTD). Such information is relevant to compute the appropriate API. A PNTD will automatically be generated by the framework we propose, from a given metamodel, either the core metamodel or one of its extensions.

The PNML description of the core metamodel and a model referring to it, is based on and fully compliant with *basic PNML* schema RELAX NG implementation [10]. We consider the core metamodel as being the P/T net metamodel. For this demonstration we will not take into consideration *structured PNML* that allows definition of Petri Net features for its structuring into several pages. We also consider the Conventions document<sup>3</sup>, the purpose of

---

<sup>2</sup>We will provide definitions for High-Level Petri Nets and Well-Formed Petri Nets in a further paper.

<sup>3</sup><http://www.informatik.hu-berlin.de/top/pnml/conventions.html>

which is to set up a reference document for labels definition. New labels definitions for new types of Petri nets should be included into the conventions document.

Extensions to the basic core metamodel will lead to PNTDs definitions that will be submitted to enrich the existing one, namely P/T net PNTD.

### 4.3 The API production chain

Let us detail in Figure 3 the production chain initially sketched in Figure 1. There are three levels:

- *ECORE* is at the MOF level. It is an Eclipse Modelling Framework implementation (EMF) of MOF 2.0 to structure any metamodel.
- *PetriNetDoc* is the entry point of our production chain. It is based on *ECORE* since we use Eclipse and represent a type of Petri net,
- *Petri net model* corresponds to the model to be processed by generated APIs.

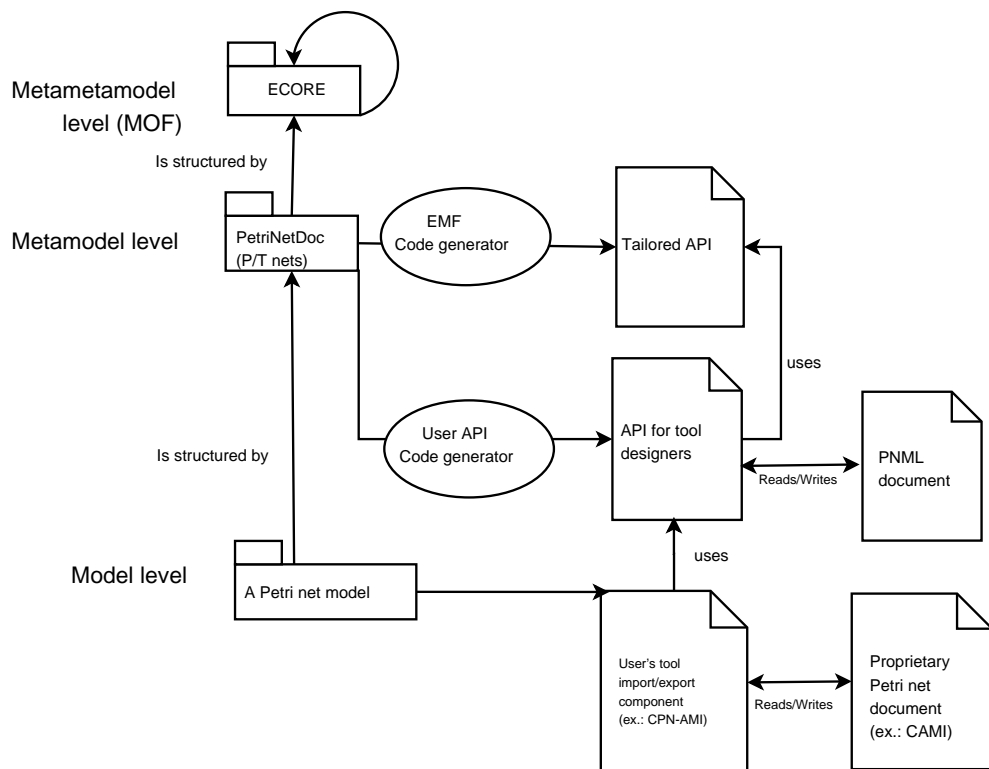


Figure 3: API production chain.

EMF allows to generate a model manipulation API known as *tailored API*. It enables application designers to create, modify, load and store models corresponding to the structure specified by their metamodel. However, this API is at too low a level to use in practise by Petri net tools designer and requires a deep knowledge of EMF. Consequently, we generate

a higher level API (called *PNAPI*) dedicated to Petri net tool designers. It provides easier to use services for model manipulation.

This PNAPI contains the following main functions:

- load/store functions that perform a translation of an internal memory representation from/to a PNML text file,
- reading primitives that allow to extract information on the Petri Nets contained in the PNML document,
- writing primitives that allow to build a Petri net using the internal representation (prior to storing it in PNML).

Using the PNAPI, tool designers should be able to quickly import and export PNML specifications into/from their proprietary Petri net representation format. It is of interest since the produced PNML is usually quite complex and thus difficult to write manually. We thus meet one of the main goals of PNML: allow for interoperability between Petri net tools designed by the community.

Our approach has the following advantages:

- our production chain relies on Eclipse which is one of the most popular portable environments ;
- even if Eclipse only handles Java, this is not a problem since this language is fully portable over operating systems. Moreover, the main objective of this API is to write translators from PNML to tool-specific representation and vice versa. If necessary, it remains possible to target other languages by rewriting the API generator ;
- we manage our production chain to produce packages that can be executed independently of Eclipse ;
- PNML specific needs (load and store in PNML format instead of XMI, for instance) could be designed and maintained easily when PNML specification changes, thanks to extraction and writing rules provided independently of EMF tailored API ;
- the model-driven approach to design the PNAPI caters for both its productivity and perennality.

## 5 The Petri net metamodel

This section presents the structure of our Petri net metamodel. We first provide information on the Core Metamodel prior to illustrating the extension mechanism.

### 5.1 The core metamodel

Let us first introduce the Petri Net Core Metamodel, PNCM. It is based on the PNML Core Model and P/T Type Model, as specified in [7].

For model engineering purposes, it is slightly different from the Core Model specified in [7]. However, it remains fully compliant ; some specificities required for computation were added. From our point of view, PNCM is the basic metamodel from which nothing could be removed, since P/T nets are the starting point for the Petri Net concept.

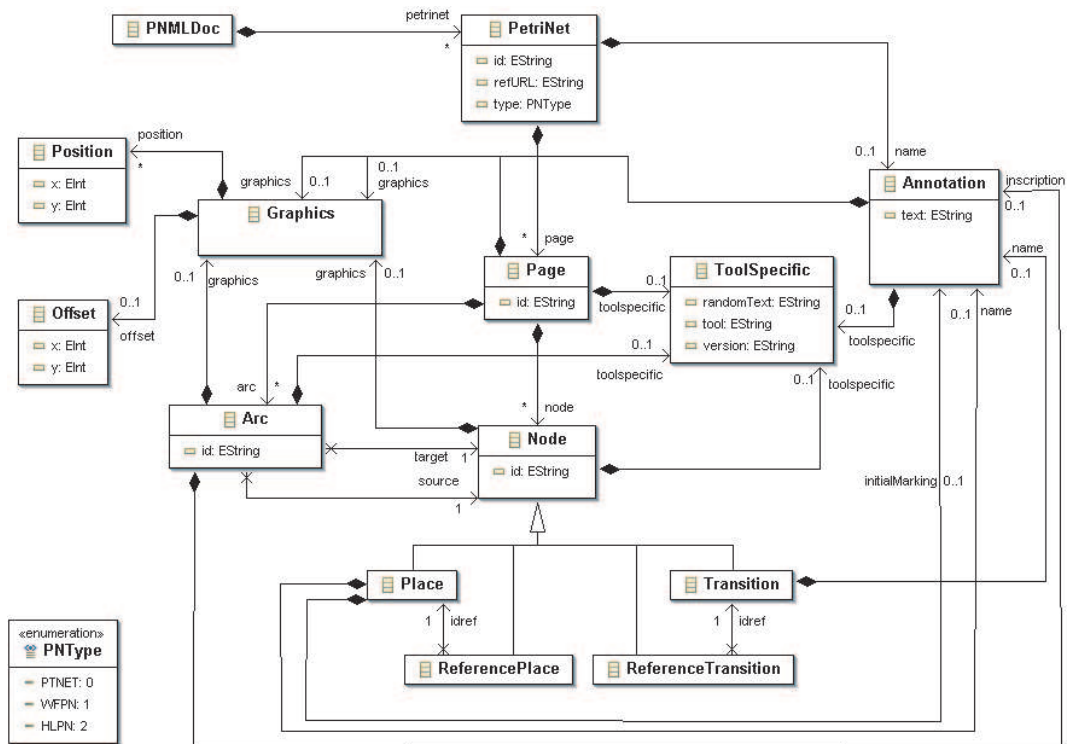


Figure 4: Petri Net PNML core metamodel

Extensions could be provided by the user. Formal extensions for High-Level Petri Nets and Well-Formed Petri Nets metamodels shall be provided, as they are in a standardisation process by ISO.

Figure 4 shows all the basic features of P/T nets<sup>4</sup>. Models that are built according to this core metamodel shall be compliant with RELAX NG based normative grammar for PNML, PNTDI and Features Definition Interface.

The proposed Core Metamodel is *minimal* for P/T nets. As mentioned before, it is based on the Core Model proposed in [7] but introduces some restrictions to make it suitable to target model productivity.

The main features of the PNCM are the following :

- the *PNMLDoc* concept is the entry point to the metamodel. It represents a PNML document which could contain one or more Petri nets. A PNML document is an XML-based file which starts with a `<pnm1>` tag and ends with `</pnm1>`.
- *PetriNet* represents a single Petri net. It contains at least one page. This is one of the first restrictions we have introduced in the PNCM to make it computationally productive. A Petri Net has a unique id, a type which specifies the class of Petri nets it belongs to, e.g. P/T nets or High-Level Petri Nets etc. This is a quite simple way to add relevant information to help checking a model w.r.t. legal principles admitted by its Petri net class. The RefURL attribute locates its PNTD.

<sup>4</sup>Note that graphical information is here considered as a basic feature of all Petri Nets classes.

- a *Page* contains Arcs and Nodes.
- the *Arc* concept represents arcs. They are not considered as *Objects* any more since this latter concept has been removed.
- *Place* and *Transition* are *Nodes*.
- the *ReferencePlace* and *ReferenceTransition* are also *Nodes* and refer directly, to *Place* and *Transition* concepts, respectively.
- All objects introduced above have either a name or an initial marking, or else an inscription etc., which is represented by means of the *Annotation* concept. Since this metamodel is minimal, it is not explicitly said that an initial marking must be an integer. In *Annotation*, one can notice that *text* attribute is generally speaking a *String*, since a name is also an *Annotation*. This is done on purpose to show that even if in the grammar specified in [7], an initial marking is a *NonNegativeInteger*, it does appear in the final PNML document surrounded by `<text>` and `</text>` tags. This may lead to misunderstanding and should be discussed.
- All these objects, including their *Annotation*, have graphical information, represented by means of the *Graphics* concept.

This metamodel was created using *Eclipse Modelling Framework* [1], a MOF implementation from Eclipse Foundation, and *Omondo free edition plugin* [2] for its graphical representation as a UML class diagram.

For practical purposes, we do not provide detailed information for the *Graphics* class.

## 5.2 The extension mechanism

As stated in section 2, we consider two types of variations in the metamodel. Minor ones mainly concern small additions such as the management of a specific object label by a given tool. Even if it is possible to reflect it at the metamodel level, it is too big a change. A simpler mechanism is then to use dedicated primitives to achieve this at the API level.

Major variations are also to be considered. They deal with semantic modifications such as the addition of inhibitor arcs or immediate transitions. Such modifications must be reflected in the metamodel since they impact semantics.

Creating a new type of places or transitions (such as FIFO places) can be achieved by deriving a new *Node* class (see figure 4). For new types of arcs (such as an inhibitor arc), new labels distinguish this specific class, according to clause 5.2.2.3 in [7]<sup>5</sup>. Creating a new kind of marking to place, such as *capacity*, could be achieved by simply adding a new attribute of *Annotation* type to *Place* concept (see figure 4).

## 6 Code generation and APIs

Code generation produces PNAPIs to manipulate standardised Petri nets using several programming languages. In this section, we show the structure of an API and briefly present how model manipulation can be performed. Examples of generated APIs are provided in Java.

---

<sup>5</sup>In our opinion, the extension mechanism for nodes and for arcs should be unified.

## 6.1 Architecture of the PNAPI

The generated PNAPI is structured as presented in Figure 5. A first module handles a neutral and internal representation suitable to support any type of Petri net (and not suitable to provide optimised representations). Based on this component, three interfaces provide specific services such as: load/store (mapping between the internal representation and PNML), modification of the Petri net model and manipulation (read-only) of the Petri net model.

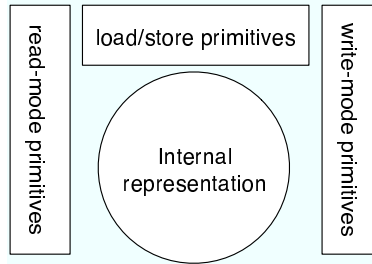


Figure 5: Structure of a generated PNAPI

The purpose is to provide a set of appropriate functions to help the tool designer with providing interoperability with the exchange format. It is important to point out that the generated PNAPI concerns a particular type of Petri Net. Thus, it is possible to check conformance of the type of Petri net manipulated. For example, one could want to create an arc between two places in a P/T net. This inconsistency would be discovered and raise an error<sup>6</sup>.

## 6.2 Load/Store primitives

Loading and storing Petri net models are based on the read and write modes. Hence primitives from these services require read/write service primitives.

```
public boolean loadPnmlModel(String pnmlDocname);

public boolean savePnmlModel(String pnmlDocname);
```

Let us consider that a tool designer wants to store the diameter of places as graphical information. He/she can do it without any modification to the metamodel by adding custom labels. The information is saved as PNML extra labels and retrieved by the load function ; this label can be used if needed.

When the PNML specification does not fit the associated PNTDI, the loader ignores irrelevant information and sends a warning to the invoker (by means of exceptions or return values according to the language capabilities). Such a practise is possible but not recommended.

## 6.3 Write-mode primitives

Let us consider the Petri Net Core Metamodel presented in section 5.1. We want to support the creation of P/T net models and its basic elements. Provided all mappings and metamodel transformations, and the PNTDI produced, functions for the write-mode purpose can

<sup>6</sup>Note that in the PNTDI, there is no rule for this purpose yet.

be automatically generated. In this code, the parameters types are the metamodel classes (model, places, transitions, arcs, etc.).

```
public boolean createPTDoc(PnmlPackage pnmldoc);

public boolean createPTNet(PetriNetClass pnc);

public boolean createPTPlace(Place pl);

public boolean createPTTransition(Transition tr);

public boolean createPTArc(Arc ac);

public boolean finalizePTDoc(String errormessage, PnmlPackage pnmldoc);
```

Creation of a PTDoc (a P/T PNML description in our example) is quite simple and requires only the file name. According to the type of Petri Net specified in the metamodel, the functions signatures vary in their naming convention: **PT** for P/T nets, **HLPN** for High-Level Petri Nets, etc.

The type of a newly created Petri net is implicit: it is the one of the PNAPI used (and thus the originating metamodel).

No verification is automatically performed when creating an object. The `finalizePTDoc` primitive processes all checks, when invoked, on a model which is assumed to be complete.

## 6.4 Read-mode primitives

Manipulation of a PNML specification occurs in the internal representation issued from a valid PNML document (by means of the `loadPnmlModel` primitive). In order to perform these operations, tool designers should use the following functions.

```
public boolean getPTNet(PetriNetClass pnc);

public boolean getPTPlace(Place pl);

public boolean getPTTransition(Transition tr);

public boolean getPTArc(Arc ac);

public boolean closePTDoc(PnmlPackage pnmldoc);
```

The PNAPI provides for Petri net objects only simple fetch services similar to those of sequential files. It is then possible to retrieve an object (place, arc or transition in our example) and then access its labels (the corresponding primitives are not shown here by lack of space). Let us recall that the PNAPI should not replace the optimised internal representation of a tool.

A Petri net model manipulation main program may have the following profile:

```
public static void main(String[] args) {

try {
    String pnmlfileName = "test_pnml.pnml";
    String errormessage = new String("");
    ManipulationAPI ma = new ManipulationAPI();
    //load the model file
    ma.loadPnmlModel(pnmlfileName);
    //create a workspace Package and PetriNetClass of "P/T type"
    PnmlPackage pnpack = new PnmlPackage();
```

```

        //The following instantiates a P/T net
PetriNetClass pnc = new PetriNetClass("ptNetb");
        //load the PNML document
ma.openPNMLDoc(pnpack);
        //get the loaded model into the internal representation
        //here, if the Petri net type does not match,
        //a warning message will be issued and/or error raised.
pnpack.getPTNet(pnc);
        //create a new place
Place pl = new Place();
        //set the name for this place
pl.setName("P1");
        //etc.,
.....
        //insert the place into the net
pnc.createPTPlace(pl);
        //finalise the Petri net (with relevant verification)
        //and close the PNML document
ma.finalizeDoc(errormessage, pnpack);
ma.closePTDoc(pnpack);
        // save the model
ma.savePnmlModel(pnmlfileName);
} catch(Exception e) {
    e.printStackTrace();
}
}

```

With such an approach, various operations could be performed as required by the tool designer. He/she will typically directly translate the PNAPI internal representation into the optimised one of his/her tool.

## 6.5 Applying the extension mechanism to an example

We are now going to exemplify our model-engineering technique with a small application. Let consider the net in figure 6, depicted as a simple P/T net having P2 as a place with capacity 3, and initial marking 2. Imagine the corresponding PNML document, generated by a P/T net handling tool, is to be sent (via email) in order to be used by another tool. Both tools handle 3D Petri net drawings. Then, there is a choice:

1. either extending the PNCM graphics subclasses (e.g, cartesian coordinates and dimension classes shown in appendix ??) to produce an enhanced API ; or
2. extend PNCM with a **tool specific** class ; or
3. directly use the API to add extra labels to model places, transitions and arcs.

In such a case, the first and second solutions are the most appropriate but heavy to deploy. The third solution could be considered since the semantics of the Petri net type remains unchanged.



Figure 6: Petri net with place capacity

Let us now suppose that we want to support place capacities in our Petri nets and share this feature with the peer tool previously mentioned. In that configuration, the first solution is the only possible one since the Petri net semantics is modified (major modification as outlined in section 2). Thus, a new Annotation attribute (PTCapacity) is created in the Metamodel, and set into the Place concept.

The Feature Definition Interface is therefore enriched with the following information:

```
<define name="PTCapacity">
  <a:documentation>
    Label definition for a user given identifier of an element describing
    its meaning.
    <contributed>Fabrice Kordon</contributed>
    <date>2005-04-10</date>
  </a:documentation>
  <element name="capacity">
    <ref name="nonnegativeintegerlabel.content"/>
  </element>
</define>
```

In addition, the PNTD is extended as follows:

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

The complete PNML representation of this example is provided in appendix A.

## 7 Conclusion

Part 2 of ISO/IEC 15909 standard aims at defining a tool interchange format for Petri nets, PNML. In this paper, we have presented an innovative way to handle a programming environment associated with ISO/IEC 15909 in order to help Petri net tools designers to use it. This implies the production of standard APIs from a metamodel, which:

- enforces a strong relation between the standard and corresponding APIs as well as versatility over the numerous Petri net types that are used by the community ;
- maintain compatibility between Petri net types and thus allow model exchanges between tools ;
- associate the standard with model engineering techniques (used in MDA) and thus freely provide them for engineers willing to apply them to Petri nets.

We have tried to deal with choices made in the current ISO/IEC 15909-2 working document. However, it turned out that some choices may have to be reconsidered, due to the new goal of reusing model engineering techniques.

To assess our proposal, we used our metamodel proposal to generate APIs to manipulate P/T nets by means of PNML. Based on the generated APIs, a simple translator from CPN-AMI [3] P/T internal format to PNML and vice-versa has been implemented. In addition to the

feasibility assessment, we could see in practise how easy it is to elaborate a translator from a tool internal format to PNML and vice versa.

We think that our design approach to the PNML standard would enforce both a safe adaptability and usability of the interchange format. This is a key to the standards being used by the community (in universities as well as in industries).

## References

- [1] *Eclipse Modeling Framework*. <http://www.download.eclipse.org/tools/emf/scripts/home.php>.
- [2] *Omondo, The Live UML Company*. <http://www.omondo.com>.
- [3] *The CPN-AMI Home page*. <http://www.lip6.fr/cpn-ami>.
- [4] 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.
- [5] X. Blanc, M.P. Gervais, and P. Sriplakich. Model Bus : Towards the Interoperability of Modelling Tools. In *2nd Workshop on MDA (MDAFA'04)*, June 2004.
- [6] ISO/IEC. Systems Engineering - Modeling Languages - High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, International Standard ISO/IEC 15909.
- [7] Ekkart KINDLER. High-level Petri Nets - Transfert Format - Working Draft for the International Standard ISO/IEC 15909 Part 2 - Version 0.5.0, November 2004.
- [8] OMG. *MetaObjectFacility Specification, document no:omg/2002-04-03*. OMG, April 2002.
- [9] OMG. MDA Guide Version 1.0.1, document no: omg/2003-06-01, 2003.
- [10] Michael WEBER. *Petri Net Markup Language*. <http://www.informatik.hu-berlin.de/top/pnml/pnml.html>.

## A Potential PNML representation for the model in figure 6

Here is a potential PNML representation for the model shown in figure 6.

```
<pnml xmlns="http://www.informatik.hu-berlin.de/top/pnml/ptNetb">
  <net id="n1" type="http://www.informatik.hu-berlin.de/top/pntd/ptNetb">
    <name>
      <text>P/T net with capacity</text>
    </name>
    <place id="p1">
      <graphics>
        <position x="23" y="49"/>
      </graphics>
      <name>
        <text>P1</text>
      </name>
      <initialMarking>
        <text>1</text>
      </initialMarking>
    </place>
    <place id="p2">
      <graphics>
        <position x="281" y="49"/>
      </graphics>
      <name>
        <text>P2</text>
      </name>
      <capacity>
        <text>3</text>
      </capacity>
      <initialMarking>
        <text>2</text>
      </initialMarking>
    </place>
    <transition id="t1">
      <graphics>
        <position x="159" y="43"/>
      </graphics>
      <name>
        <text>T1</text>
      </name>
    </transition>
    <arc id="a1" source="p1" target="t1">
      <graphics>
        <position x="89" y="60"/>
      </graphics>
    </arc>
    <arc id="a2" source="t1" target="p2">
      <graphics>
        <position x="217" y="60"/>
      </graphics>
    </arc>
  </net>
</pnml>
```