# Extending PNML Scope: the Prioritised Petri Nets Experience

Lom-Messan Hillah[1], Fabrice Kordon[2], Charles Lakos[3], and Laure Petrucci[4]

[1] LIP6, CNRS UMR 7606
and Université Paris Ouest Nanterre La Défense
200, avenue de la République, F-92001 Nanterre Cedex, France
`Lom-Messan.Hillah@lip6.fr`
[2] Université P. & M. Curie LIP6 - CNRS UMR 7606
4 Place Jussieu, F-75252 Paris cedex 05, France
`Fabrice.Kordon@lip6.fr`
[3] University of Adelaide, Adelaide, SA 5005, Australia
`Charles.Lakos@adelaide.edu.au`
[4] LIPN, CNRS UMR 7030, Université Paris XIII
99, avenue Jean-Baptiste Clément, F-93430 Villetaneuse, France
`Laure.Petrucci@lipn.univ-paris13.fr`

**Abstract.** The Petri net standard ISO/IEC 15909 comprises 3 parts. The first one defines the most used net types, the second an interchange format for these — both are published. The third part deals with Petri net extensions, in particular structuring mechanisms and the introduction of additional, more elaborate net types within the standard.
This paper focuses on the latter issue: how should a new net type be added, while guaranteeing the compatibility with the current standard. The extension of Petri nets with static or dynamic priorities is studied, showing design choices to ensure the desired compatibility. The result is integrated within the standard companion tool, PNML Framework. Then, the approach is generalised so as to be used at a later stage for other Petri nets extensions.

**Keywords:** Standardisation, PNML, Prioritised Petri Nets

## 1 Introduction

The International Standard on Petri nets, ISO/IEC 15909, comprises three parts. The first one (ISO/IEC 15909-1) deals with basic definitions of several Petri net types: Place/Transition, Symmetric, and High-level nets. It was published in December 2004 [5].

The second part, ISO/IEC 15909-2, defines the interchange format for Petri net models: the Petri Net Markup Language [7] (PNML, an XML-based representation). This part of the standard was published on February 2011 [6]. It can now be used by tool developers in the Petri Nets community with, for example, the companion tool to the standard, PNML Framework [4].

The standardisation effort is now focussed on the third part. ISO/IEC 15909-3 aims at defining enrichments and extensions on the whole family of Petri nets.

Extensions are, for instance, the support of modularity, time or probabilities. Enrichments consider less significant semantic changes such as inhibitor arcs, capacity places, etc. This raises flexibility and compatibility issues in the standard.

One of the interesting features in Petri nets is priorities. There are a number possibilities: static priorities and dynamic priorities, as summarised in [9]. Since such characteristics are of interest for several classes of Petri nets (from P/T up to high-level), it is highly desirable to investigate their definition in an orthogonal way that can be associated with any of the existing Petri net types.

This paper focuses on this objective: enrich existing Petri net types with both static and dynamic priorities. To do so, we explore a modular and generic enrichment mechanism that benefits from the current metamodels architecture of the standard. Thus, we can preserve consistency between existing Petri net types and those obtained with the proposed enrichments.

The paper is structured as follows. Section 2 summarises the specific aims of part 3 of the standard. Section 3 defines prioritised Petri nets, before describing, in Section 4, the introduction of Prioritised Petri nets types in the standard metamodeling framework. The metamodels obtained are then integrated within PNML Framework, and experimental results are reported in Section 4.3. Section 5 discusses the expertise drawn from the case of Prioritised Petri nets, so as to give general guidelines for the integration of a new Petri net type within the standard.

## 2 Aims of ISO/IEC 15909-3

While parts 1 and 2 of the ISO/IEC 15909 standard address simple and common Petri nets types, part 3 is concerned with extensions. These can take several forms, as described in Section 2.1. The work on these issues started with a one-year study group drawing conclusions w.r.t. the scope to be addressed. According to the study group conclusions, the standardisation project was launched in November 2010, for delivery within 5 years. The choices to be made must of course ensure compatibility with the previous parts of the standard, as discussed in Section 2.2.

### 2.1 Petri nets extensions

The Petri net extensions considered can be of different kinds: nodes or arcs extensions, structuring mechanisms, new Petri net types. One can even consider the possibility of tools exchanging Petri net properties through the net files. In this section, we present the main ideas underlying these possibilities.

*Enrichments* are concerned with the addition of a new type of node or arc to an already existing Petri net type. Typical examples of such extensions are inhibitor arcs or capacity places.

Enrichments[5] are rather simple extensions since, although they modify the net semantics, they do not not require the manipulation of data which is not already described in the Petri net type. Indeed, let us illustrate this with the capacity place example. If a net is extended with capacity places, the capacity only indicates a maximum marking the place can hold. The marking being already an element of the Petri net type description, adding capacities is straightforward. The mechanism for doing so is independent of the Petri net type: it is defined as a maximal marking for the place in the same manner as its initial marking is defined.

We have already tested enrichment mechanisms in practice within PNML Framework, the companion tool to the standard [11, 4], by introducing special arcs such as inhibitor, test, and reset arcs. As expected, this experiment proved successful. An extension of the PNML (Petri Net Markup Language) grammar for these special arcs is available online at [7].

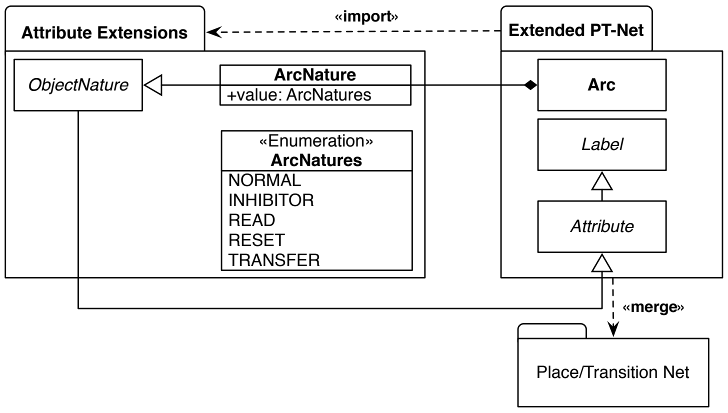

**Fig. 1.** Extending PT-Net with special arcs.

The experiment consisted in extending the metamodel of PT-Net, first by defining the special arcs nature as attribute extensions, whose metamodel is depicted in Fig. 1. Then, the extended PT-net metamodel is built by merging the current PT-Net metamodel and importing the attribute extensions one. This modular definition approach is put into practice in Section 4 and the use of the `import` and `merge` relationships explained.

*Modularity and structuring mechanisms* are essential for modelling and analysis purposes. Such mechanisms are independent of the Petri net type and should thus be general enough. Preliminary theoretical work in that direction has been presented in [8]. The main features are the following:

[5] The name chosen is consistent with the notion of enrichment for abstract data types [3].

– each module is composed of an interface and an implementation,
– the implementation is a Petri net,
– interfaces import and export Petri net elements (according to the implementation Petri net type): places, transitions, data types and operations,
– modules can be instantiated and connected so as to constitute an actual complex system.

Even though the work on structuring mechanisms has progressed well, there are still numerous issues to be considered, as detailed in [8]. For example, the semantics for connecting modules can vary, and node fusion policies could be defined. This leads to a more elaborate extension of Petri net types. Moreover, practical experiments still need to be conducted.

*New Petri net types* build on existing types — at least the Core Petri Net model — enhancing them with specific features which could be a new attribute or a new element. However, they are distinguished from "enrichments" in that they necessitate the elaboration of specific additional constructs. As an example, dynamically prioritised Petri nets associate with each Petri net transition a priority function with a marking as input and some value, e.g. a real number, as output. In this case, markings are already part of the defined Petri net elements, but real numbers are not. Therefore, in contrast to the capacity place example discussed above, dynamic priority is not an enrichment.

The core of this paper concerns how to introduce new Petri net types, and will thus be detailed by first studying the case of prioritised Petri nets in Section 4, and then generalising the approach in Section 5 to give guidelines for introducing new Petri net types in the future.

*Properties* such as safety and liveness properties, are a much longer term issue for standardisation, and will certainly not be achieved in the first release of part 3 of the standard, but might be in a future revision. The idea is to define storage mechanisms for properties so as to include properties within Petri net files. Thus, properties could be computed by one tool and later be exploited by another one.

### 2.2 Compatibility issues

The new features brought by part 3 of the standard must of course ensure compatibility with the previous stages, i.e. parts 1 and 2. This is essential since the already defined Petri net types constitute the building blocks.

Hence, an enrichment, such as the aforementioned capacity places, basically involves the addition of a new attribute to an already existing class of objects. In the case of capacity places, this attribute is a maximum marking. Note that this kind of extension easily applies to any type of net: a maximum marking is a marking as defined for the net type, be it a P/T net or a high-level net.

Similarly, the modular or structuring constructs must apply to all kinds of nets. They define the different parts of a module, that is the interface and the implementation, the composition policy, etc. The implementation is then an already known net type and the interface adds attributes telling which objects are imported or exported. The composition policy has so far been quite simple: place fusion and transition fusion, essentially. Nonetheless, work in this area must be pursued further.

Finally, adding new Petri net types will, as discussed in Section 2.1, build on at least the Petri net core model, thus preserving the essential characteristics common to all Petri net types. However, one can easily imagine that Prioritised P/T nets build on the P/T nets model while Prioritised high-level nets build on high-level nets. Further, several kinds of extensions could be applied so as to obtain a more elaborate net type, e.g. Prioritised Modular High-Level nets. Therefore, extensions must be carefully designed, allowing for a high degree of compatibility.

An important point must be investigated while some characteristics of Petri Nets can be considered as "orthogonal" (i.e. without influence on each other). As an example, priorities and colours can be considered separately and combined together because they do not affect the same attributes[6]. Such an orthogonality is important in the design of new Petri net types in the standard. This is discussed further in section 5.

## 3 Prioritised Petri Nets

This section introduces the definition of prioritised Petri nets, starting with static priorities.

**Definition 1 (Statically Prioritised Petri net).**
*A* Statically Prioritised Petri net *is a tuple SPPN* $= (P, T, W, M_0, \rho)$*, where:*

- $(P, T, W, M_0)$ *is a Petri net.*
- $\rho$ *is the* static priority function *mapping a transition into* $\mathbb{R}^+$.

We can also consider the case where the priority of transitions is *dynamic*, i.e. it depends on the current marking [1]. This definition was introduced in [9]. Note that the only difference with statically prioritised Petri nets concerns the priority function $\rho$.

**Definition 2 (Prioritised Petri net).**
*A* Prioritised Petri net *is a tuple PPN* $= (P, T, W, M_0, \rho)$*, where:*

- $(P, T, W, M_0)$ *is a Petri net.*
- $\rho$ *is the* priority function *mapping a marking and a transition into* $\mathbb{R}^+$.

The behaviour of a prioritised Petri net is now detailed, markings being those of the associated Petri net. Note that the firing rule is the same as for non-prioritised Petri nets, the priority scheme influencing only the enabling condition.

[6] for shared attributes like marking, we so far duplicate them. As an example, there are PTMarking for P/T nets and HLMarking for high-level nets.

**Definition 3 (Prioritised enabling rule).**

– *A transition* $t \in T$ *is* priority enabled *in marking* $M$*, denoted by* $M[t\rangle^\rho$*, iff:*
  - *it is enabled, i.e.* $M[t\rangle$*, and*
  - *no transition of higher priority is enabled, i.e.* $\forall t' : M[t'\rangle \Rightarrow \rho(M, t) \geq \rho(M, t')$*.*

– *The definition of the priority function* $\rho$ *is extended to sets and sequences of transitions (and even markings* $M$*):*
  - $\forall X \subseteq T : \rho(M, X) = \max\{\rho(M, t) \mid t \in X \wedge M[t\rangle\}$
  - $\forall \sigma \in T^* : \rho(M, \sigma) = \min\{\rho(M', t') \mid M'[t'\rangle^\rho$ *occurs in* $M[\sigma\rangle^\rho\}$*.*

In the definition of $\rho(M, X)$, the set $X$ will often be the set $T$ of all transitions, in which case the $T$ could be omitted and we could view this as a priority of the marking, i.e. $\rho(M)$. The definition of $\rho(M, X)$ means that we can write the condition under which transition $t$ is priority enabled in marking $M$ as $M[t\rangle^\rho$, or in the expanded form $M[t\rangle \wedge \rho(M, t) = \rho(M, T)$. We prefer the latter form if the range of transitions is ambiguous.

If the priority function is constantly zero over all markings and all transitions, then the behaviour of a Prioritised Petri Net is isomorphic to that of the underlying Petri Net.

Note that we choose to define priority as a positive real-valued function over markings and transitions — the higher the value, the greater the priority. We could equally define priority in terms of a *rank function* which maps markings and transitions to positive real values, but where the smaller value has the higher priority. This would be appropriate, for example, if the rank were an indication of earliest firing time. Note that the dependence of the priority function on the markings (as well as the transitions) means that *the priority is dynamic.*

## 4 Adding Prioritised Petri Nets to the Standard

This section introduces the Petri nets metamodels modular definition approach defined in Part 2 of the standard and how we put it into practice to design prioritised Petri nets.

### 4.1 Current Metamodels Architecture

Figure 2 shows an overview of the metamodels architecture currently defined in Part 2 of the standard. This architecture features three main Petri net types: Place/Transition, Symmetric and High-level Petri nets. They rely on the common foundation offered by the PNML Core Model. The PNML Core Model provides the structural definition of all Petri nets, which consists of nodes and arcs and an abstract definition of their labels. There is no restriction on labels since the PNML Core Model is not a concrete Petri net type.

Such a modular architecture favours reuse between net types. Reuse takes two forms in the architectural pattern of the standard: package `import` and `merge` relationships, as defined in the UML standard [12].

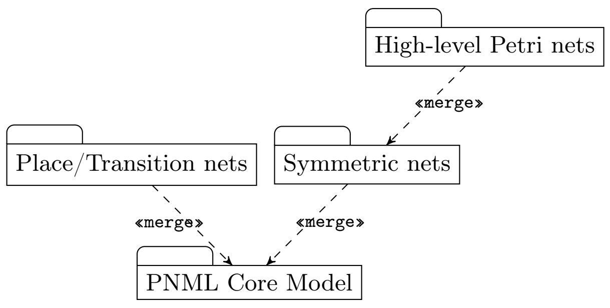

**Fig. 2.** Metamodels architecture currently defined in ISO/IEC-15909 Part 2.

`Import` is meant to use an element from another namespace (package) without the need to fully qualify it. For example, when package `A` includes: `import B.b`, then in `A` we can directly refer to `b` without saying `B.b`. But `b` still belongs to the namespace `B`. In the ISO/IEC 15909-2 standard, Symmetric nets import sorts packages such as `Finite Enumerations, Cyclic Enumerations, Booleans`, etc.

`Merge` is meant to combine similar elements from the merged namespace to the merging one. For example, let us assume that `A.a`, `B.a` and `B.b` are defined. If `B` is merged into `A` (`B` being the target of the relationship), it will result in a new package name `A'`:

- all elements of `B` now explicitly belong to `A'` (e.g., `A'.b`);
- `A.a` and `B.a` are merged into a single `A'.a` which combines the characteristics of both;
- actually, since `A` is the merging package (or the receiving package), `A` becomes `A'` (in the model, it is still named `A`).

`Merge` is useful for incremental definitions (extensions) of the same concept for different purposes.

In the standard, this form of reuse is implemented for instance by defining Place/Transition nets upon the Core Model and High-level nets upon Symmetric nets, as depicted in Figure 2. That is why Symmetric nets elements and annotations are also valid in High-Level Petri nets (but not considered as Symmetric nets namespace elements anymore).

This extensible architecture is compatible with further new net types definitions, as well as with orthogonal extensions shared by different net types. These two extension schemes will be put into practice for defining prioritised Petri nets, as discussed in the next section.

## 4.2 Metamodels for PT-Nets with Priorities

A prioritised Petri net basically associates a priority description with an existing standardised Petri net, thus building a new Petri net type. The metamodel in

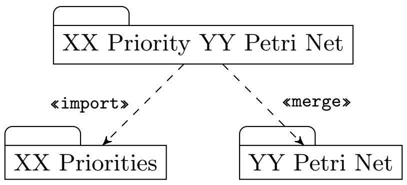

**Fig. 3.** Modular construction of prioritised Petri Nets metamodels.

Figure 3 illustrates this modular definition approach. It shows a blueprint for instantiating a concrete prioritised Petri net type, by merging a concrete Petri net type and importing a concrete priority package. The `XX Priority` package is the virtual representation of a concrete priority package and the `YY Petri net` is the virtual representation of a concrete Petri net type.

For example, Figure 4 shows a prioritised PT-Net using static priorities only. It is built upon a standardised PT-Net which it merges, and a `Priority Core` package, which it imports. The `Priority Core` package provides the building blocks to define `Static Priorities`, as depicted by Figure 5.

The purpose of the `Priority Core` package is to provide :

– the root metaclass for priorities, represented by the `Priority` metaclass;
– a priority level, which is an evaluated value represented by `PrioLevel`, associated with each instance of `Priority`;
– the ordering policy among the priority values of the prioritised Petri net. This ordering policy is represented by the `PrioOrderingPolicy` metaclass.

The purpose of priority levels is to provide an ordered scalar enumeration of values such that either the higher the value, the higher the priority, or the lower the value, the higher the priority. With the `Priority Core` package, and thanks to the `PrioLevel` metaclass, static priorities can thus be attached to transitions, as in the `Static Priority PT-Net` shown in Figure 4.

Using the same approach, Figure 6 shows a prioritised PT-Net which uses dynamic priorities. Dynamic priorities are built upon `Priority Core`.

This modular construction follows the extension schemes adopted so far in the PNML standard, that were explained earlier in this section. For instance,

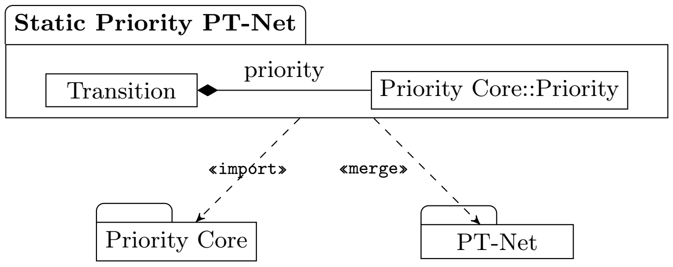

**Fig. 4.** Prioritised PT-Net metamodel showing how the priority description is attached.

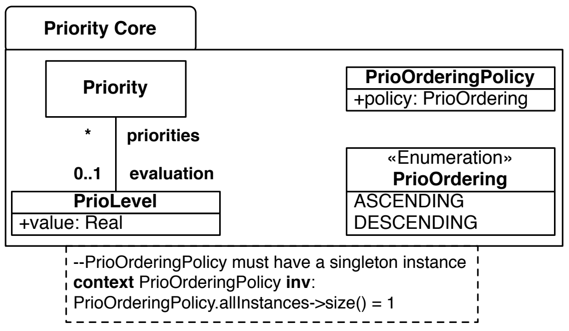

**Fig. 5.** Core package of priorities.

High-Level Petri nets build upon Symmetric nets that they merge, and new specific sorts (such as List, String and arbitrary user-defined sorts) that they import. The use of the `merge` and `import` relationships is therefore consistent.

This approach is consistent with the idea that a new Petri net type subsumes the underlying one it builds upon, but the algebraic expressions it reuses are generally orthogonal to net types. Next, we introduce the metamodel for priorities.

**Priority Metamodel** Prioritised Petri nets augment other net models (e.g. PT or Symmetric nets) by associating a priority description with the transitions. Such priority schemes are of two kinds:

– *static priorities*, where the priorities are given by constant values which are solely determined by the associated transition[7];
– *dynamic priorities*, where the priorities are functions depending both on the transition and the current net marking.

Figure 7 shows the modular architecture of priorities metamodels. The `Priority Core` package (detailed in Figure 5) provides the building blocks to define

[7] For high-level nets such as Coloured nets, the priorities are given by constant values which are solely determined by the associated binding element.

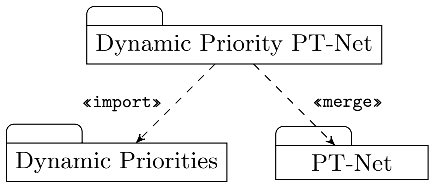

**Fig. 6.** Prioritised PT-Net metamodel using dynamic priority

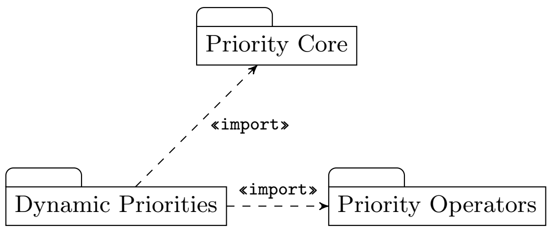

**Fig. 7.** Metamodel for priorities.

both `Static Priorities` and `Dynamic Priorities`. However, dynamic priorities are further defined using `Priority Operators`. Dynamic priorities can encompass static ones by using a constant function (for the sake of consistency in the use of priority operators).

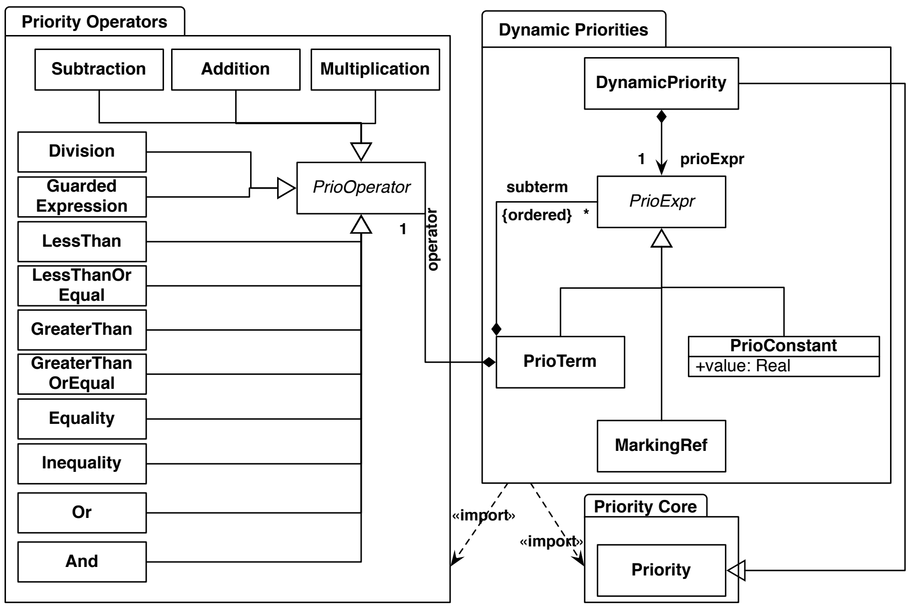

**Fig. 8.** Dynamic priorities and priorities operators packages.

Figure 8 shows how the `Dynamic Priorities` metamodel is built. A `Dynamic-Priority` *is a* `Priority Core::Priority`. It contains a priority expression (`PrioExpr`). A concrete priority expression is either a `PrioTerm` which represents a term, a `PrioConstant` which holds a constant value or `MarkingRef` which will hold a reference to the marking of a place.

Note that the actual reference to the metaclass representing markings is missing. It must be added as an attribute (named `ref`) to `MarkingRef` once the concrete prioritised Petri net type is created. Its type will then be a reference to the actual underlying Petri net type marking metaclass. For instance, in the case of prioritised PT-Net, this `ref` attribute will refer to the `PTMarking` metaclass.

A `PrioTerm` is composed of an operator (`PrioOperator`) and ordered subterms. This definition enables priority expressions to be encoded in abstract syntax trees (AST). For example, the conditional priority expression: `if` $M(P2) > 3$ `then` $3 * M(P2)$ `else` $2 * M(P1)$, is encoded by the AST of Figure 9, assuming that:

- P1 and P2 are places;
- M(P1) and M(P2) are respectively markings of P1 and P2;
- T1 is a transition the dynamic priority expression is attached to.

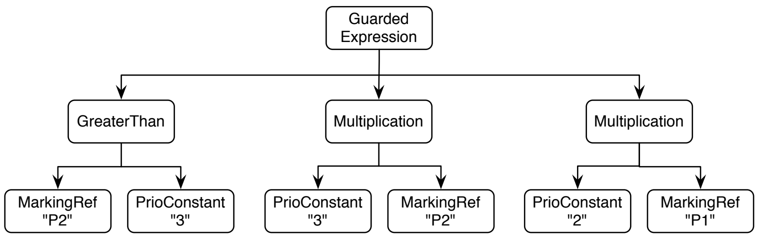

**Fig. 9.** AST of the conditional expression: `if` $M(P2) > 3$ `then` $3 * M(P2)$ `else` $2 * M(P1)$.

The priority operators are gathered within the `Priority Operators` package to allow for more flexibility in extending this priority framework. New operators can thus be added easily to this package.

Note that all these operators can also be found in ISO/IEC 15909-2, but are scattered among different sorts packages, thus directly tied to the sort they are most relevant for. We suggest for the next revision of the standard that they be gathered in separate and dedicated packages (e.g. arithmetic operators, relational operators, etc.). This refactoring will allow for more reusability across different Petri net type algebras definitions.

### 4.3 Towards experimentation with PNML Framework

PNML Framework is one of the standard's companion tools, which provides an intuitive and easy way to use Java Application Programing Interface (API) to handle standardised Petri nets models. The design and development of PNML Framework follows model-driven engineering (MDE) principles and relies on implementing mature technology such as Eclipse Modeling Framework (EMF).

Thanks to MDE, PNML Framework already uses the same modular definition approach as in the standard. Dealing with new Petri net types as proposed in this extension framework will thus be implemented in the same way as it was for the first standardised net types [4]. PNML-specific information (i.e., XML tags and their relationships) is embedded in the metamodels as annotation. The metamodels (in EMF) are thus self-contained w.r.t to PNML. This PNML-specific information can thus be captured during code generation of the appropriate reader and writer methods.

Using such an approach, an API to handle the new net type models can be generated in any output language, not only Java. This is possible because EMF format is the standardised eXtended Metadata Interchange (XMI), which is XML-based. It is thus open to any technology which can handle XML. In PNML Framework, code generation templates are designed to automatically capture these annotations.

The standard uses UML `merge` relationship to implement the reuse between metamodels. Therefore, UML `merge` constraints (preconditions) and transformations (postconditions) should also apply in our framework as well, at the design level.

When we started the development of PNML Framework as a means for early assessment of the standard design choices, EMF did provide powerful code generation capabilities (including code merging). However it did not provide models merging in the sense defined by UML, which was a disappointment. It was up to the modeler to come up with a way to implement this. UML plugin did provide models merging, but this needs a round-trip transformation from EMF to UML and vice versa, which practically turned out to be messy.

Recently, EMF Compare plugin now provides a workspace editor and a programmaging interface to compare and merge models, in a version control fashion. This environment could be used to perform a basic merger in the following main steps:

1. (a) If the source package of the merge relationship does not have any specific elements different from the merge relationship target package, duplicate the package of target Petri net type in the merge relationship and rename it to the source of the merge relationship.
   (b) If there are more than one merge relationship, perform this iteratively by pair of packages, the source being incrementally augmented.
   (c) If the source package does have specific elements, then design it first and apply the merge with one target. If there many targets, apply previous step.
2. Select the packages to import and import them.
3. Add relationships and attributes which need the merge operation to complete first.

IBM's Rational Software Architect also performs UML models comparison and merger [10] but it is not free or open software. Up to now, we let the modeler choose the means to perform the merger. Models merging is an important topic which is addressed, not only in the UML standard, but also by several studies [13, 14].

In the next section, we propose a generalisation of the metamodel modular definition approach we presented in this section, as an extension framework for the definition of new Petri net types.

## 5 Generalisation

We now generalise the approach used to create prioritised Petri nets metamodels to set up an extension framework as a proposal to be considered for ISO/IEC 15909-3. The purpose is to compose extensions on an existing Petri net type to build a new net type. It is illustrated by Figure 10, where `XX Extension` and `ZZ Extension` are extensions metamodels (e.g. priority and time) which are composed over an existing `YY Petri net` type to build the `(XX ◦ ZZ) YY Petri net` type metamodel.

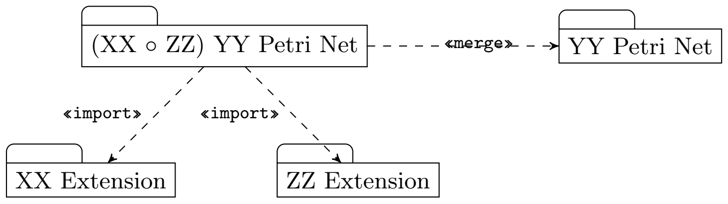

**Fig. 10.** Modular construction of a new net type by composing selected extensions.

This generalised modular definition approach involves two important characteristics to build the new net type, that are the *orthogonality* and *compatibility* of the combined features (extensions).

Orthogonality must guarantee upward compatibility: current net types definitions must have the ability to be extracted from new definitions that build upon them. For example, a Core model can currently be extracted from a Symmetric net model; a partial Symmetric net model can be extracted from a High-Level net model, after having pruned annotations and inscriptions that are not recognised in Symmetric nets.

Let us consider a Petri net type made of $n$ extensions that are orthogonal in the sense defined in section 2.2. Compatibility must guarantee that the firing rule of the new net is sound. This is the case when the set of firable transitions can be expressed as follows:

$$T_f = \bigcap_{i=1}^{n} \mathit{firing}_i(T)$$

where $T_f$ is the set of firable transitions and $\mathit{firing}_i$ are partial firing functions using the dedicated attributes associated with a Petri net type extension.

These characteristics yield semantic issues that we are currently investigating with a group of Petri nets experts to assess how they should be tackled.

Underlying issues regarding semantics of Petri nets in terms of: (*i*) abbreviation (e.g. P/T vs. Colored nets), (*ii*) extension of the modeling power (e.g. inhibitor arcs) and (*iii*) change of semantic domain (e.g. time vs. stochastic) must be properly addressed [2]. We will continuously submit the outcome of this investigative work to the ISO/IEC JTC1/SC7 WG19 working group, responsible for the standardisation of Petri nets.

Experts must therefore pay careful attention to the compatibility issue between features since it is not considered at the syntactic level which concerns their metamodels definition.

## 6 Conclusion

The standardisation effort of the International Standard ISO/IEC 15909 is currently focused on the third part, where enrichments and extensions to Petri nets are being defined. This paper presents an extension scheme based on the standard approach, in order to define prioritised Petri nets. The presented work is structured in two steps: first the formal definition of prioritised Petri nets and their enabling rule, and then their metamodels definition.

Prioritised nets metamodels are built in a modular way. First, the priority core metamodel defines the necessary concepts for static priorities. Then the dynamic priorities package reuses the core package, while adding operators from the priority operators package to its definition. Using these building blocks, a static priority PT-Net can thus be defined upon the classic PT-Net package from the standard, using the priority core. A dynamic priority PT-Net can also be defined in the same way, this time using the dynamic priorities package.

Integrating new Petri net types defined using such an approach into the companion tool, PNML Framework, is no more different than the initial work which enables the support of the current standardized types (PT-Net, Symmetric Net and High-Level Petri Net). PNML Framework being based on mature model-driven engineering tools such as Eclipse Modeling Framework, enabling support of new Petri net types follows three simple steps: (*i*) create the metamodels of the extensions and the new type, (*ii*) annotate the metamodels with PNML-specific information (XML tags and attributes), and finally (*iii*) click on a button to generate the Java API to handle the new type. The annotation step follows simple conventions that are embedded in the current metamodels, and which are easy to reproduce. Code generation templates have been designed to capture these annotations.

The purpose of this investigative work is to propose an extension framework which enables experts to easily define new Petri net types, in a consistent way with the current standard approach. Orthogonality and compatibility of combined extensions to define new Petri net types are paramount for the semantic aspect of the new net types, in particular regarding firing rules. Syntax will usually not be an issue, as the metamodel definition approach presented does not consider semantic rules.

Perspectives to this work include presenting this approach as a contribution to the next plenary session of the ISO/SC 7/WG 19 working group (responsible for the standardisation of Petri nets in the ISO/IEC 15909 series), and experiment new Petri net types definitions involving different features combination such as time and priorities.

## References

1. F. Bause. Analysis of Petri nets with a dynamic priority method. In Azéma, P. and Balbo, G., editors, *Proc. 18th International Conference on Application and Theory of Petri Nets, Toulouse, France, June 1997*, volume 1248 of *LNCS*, pages 215–234, Berlin, Germany, June 1997. Springer-Verlag.
2. M. Diaz, editor. *Petri Nets, Fundamental Models, Verification and Applications.* Wiley-ISTE, 2009.
3. J. A. Gougen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In *Current Trends in Programming Methodology*, pages 80–149. Prentice Hall, 1978.
4. L. Hillah, F. Kordon, L. Petrucci, and N. Trèves. PNML Framework: an extendable reference implementation of the Petri Net Markup Language. In *Proc. 31st Int. Conf. Application and Theory of Petri Nets and Other Models of Concurrency (PetriNets'2010), Braga, Portugal, June 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 318–327. Springer, June 2010.
5. ISO/IEC. Software and Systems Engineering - High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, International Standard ISO/IEC 15909, December 2004.
6. ISO/IEC. Software and Systems Engineering - High-level Petri Nets, Part 2: Transfer Format, International Standard ISO/IEC 15909, February 2011.
7. ISO/IEC/JTC1/SC7/WG19. *The Petri Net Markup Language home page.* `http://www.pnml.org`, 2010.
8. E. Kindler and L. Petrucci. Towards a standard for modular Petri nets: A formalisation. In *Proc. 30th Int. Conf. Application and Theory of Petri Nets and Other Models of Concurrency (PetriNets'2009), Paris, France, June 2009*, volume 5606 of *Lecture Notes in Computer Science*, pages 43–62. Springer, June 2009.
9. C. Lakos and L. Petrucci. Modular state spaces for prioritised Petri nets. In *Proc. Monterey Workshop, Redmond, WA, USA*, volume 6662 of *Lecture Notes in Computer Science*, pages 136–156. Springer, Apr. 2010.
10. K. Letkeman. *Comparing and merging UML models in IBM Rational Software Architect: Part 3 -A deeper understanding of model merging.* IBM, `http://www.ibm.com/developerworks/rational/library/05/802_comp3/`, 2005.
11. LIP6. *The PNML Framework home page.* `http://pnml.lip6.fr/`, 2011.
12. OMG. *Unified Modeling Language: Superstructure - Version 2.4 - ptc/2010-11-14*, Jan. 2011.
13. P. Sriplakich, X. Blanc, and M.-P. Gervais. Supporting transparent model update in distributed CASE tool integration. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1759–1766, New York, NY, USA, 2006. ACM.
14. B. Westfechtel. A formal approach to three-way merging of emf models. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, pages 31–41, New York, NY, USA, 2010. ACM.