# Semantic Embedding of Petri Nets into Event-B

J. Christian Attiogbé

LINA - UMR CNRS 6241
2, rue de la Houssinière, B.P.92208, F-44322 Nantes Cedex 3, France
Christian.Attiogbe@univ-nantes.fr

**Abstract.** We present an embedding of Petri nets into B abstract systems. The embedding is achieved by translating both the static structure (modelling aspect) and the evolution semantics of Petri nets. The static structure of a Petri net is captured within a B abstract system through a graph structure. This abstract system is then included in another abstract system which captures the evolution semantics of Petri nets. The evolution semantics results in some B events depending on the chosen policies: either basic nets or high level Petri nets. The current embedding enables one to use conjointly Petri nets and Event-B in the same system development, but at different steps and for various analysis.

**Keywords:** B System, Petri Nets, Embedding Techniques, Method and Tool Integration

## 1 Introduction

Reliable system development requires the use of concepts, languages, tools and methods which are provided by formal approaches. Several methods exist but are mono-paradigms. However, real size systems often overwhelm the scope covered by mono-paradigm specification techniques and their complexity requires an adequate integration of appropriate techniques and methods for both the development and the formal analysis. Current research efforts focus on the combination of various approaches and their specific tools in order to strengthen their impact on industrial system treatment. Therefore, there are some requirements to make formal methods more practical and efficient in their usage: *i)* they should be linked with *engineering practices* and techniques; *ii)* their *mechanization* by providing powerful and *operational development tools*. These points are still challenges for the formal method community and therefore they motivate our work.

The integration of various formal methods may be motivated by different kind of combinations: the complementarity of methods so as to cover the facets of the application at hand, the need of specific techniques such as composition and refinement, or specific reasoning techniques such as theorem proving and model checking, or some pragmatic considerations such as the pragmatical aspect of graphical formalisms and the interoperability of tool supports.

In the current work we study the integration of Petri nets and B in order to use conjointly both approaches in the same development. The motivation is to benefit from the complementarity of both approaches. Petri nets formalism may be used as a graphical front-end of a B development project. The B framework may follow to complement formal analysis of the system modelled using Petri nets. On the one hand, Petri nets formalism are widely used [17,19,18,14] by engineers and also in academic or research projects. Petri nets also have graphical facilities, simulation and liveness property verification facilities via powerful model checking techniques. On the other hand, B is a model-based approach which permits correct development with refinement from abstract specifications to executable codes; it is based on theorem proving technique and it offers (mainly) safety properties verification facilities.

The contribution of this article resides in *i)* the definition of a (B) generic structure to capture Petri nets models and semantics; *ii)* the means to systematically embed Petri nets structure and their evolution rules into Event-B. This leads to the development of a bridge between Petri nets and B.

The article is organised as follows: in Section 2 we introduce the Petri nets formalism and the B Systems approach. Section 3 is devoted to the stepwise embedding of Petri nets into B: basic nets are first considered and then generalized to high level nets. Section 4 gives some issues related to analysis and in Section 5 we give some concluding remarks.

## 2 Petri Nets and B Systems

### 2.1 An Overview of Petri Nets (P-nets)

Formally, a P-net is a 4-tuple $(P,\ T,\ Pre,\ Post)$ where :

- $P$ is a finite set of places , (with $\mid P \mid = m$, the cardinal of $P$);
- $T$ is a finite set of transitions, (with $\mid T \mid = n$, the cardinal of $T$);
- $P$ and $T$ are disjoint sets ($P \cap T = \{\}$);
- $Pre : P \times T \rightarrow \mathbb{N}$ is an input function, $Pre(p,t)$ denotes the number of arcs from the place $p$ to the transition $t$;
- $Post : P \times T \rightarrow \mathbb{N}$ is an output function, $Post(p,t)$ denotes the number of arcs from the transition $t$ to the place $p$.

Practically, a P-net is a bipartite directed graph whose arcs connect nodes from two distinct sets; the set of places and the set of transitions. Petri nets are equipped with a graphical formalism where the places are connected to the transitions using the directed arcs.

*Graph associated to a P-net.* The graph associated to a net $N$ is described by:

- $\Gamma_p$ the transitions reachable from each place:
  $\forall p \in P . \Gamma_p(p) = \{t \in T \mid Pre(p,\ t) > 0\}$
- $\Gamma_t$ the places reachable from each transition:
  $\forall t \in T . \Gamma_t(t) = \{p \in P \mid Post(p,\ t) > 0\}$
- $W_{in}$ the weight of each input arc: $\forall\ p \in P, \forall\ t \in T\ .\ W_{in}(p,\ t) = Pre(p,t)$ and
- $W_{out}$ the weight of each output arc: $\forall\ p \in P, \forall\ t \in T\ .\ W_{out}(p,\ t) = Post(p,t)$

The graph associated to a P-net is the abstract representation which is used to manipulate the net. The places connected to a transition with an arc from each place to the transition are the *input places* of the transition. The places connected to a transition with an arc from the transition to each place are the *output places* of the transition.

*P-net marking.* A marked net $M_N = (N, \mu)$ is made of a net $N$ and a mapping $\mu : P \rightarrow \mathbb{N}$.
$\mu(p)$ is the number of tokens within $p$; it is called the *marking* of the place $p$. The initial marking $M_0$ of a net is the n-tuple made of the initial marking of all the places $p_i$ of the net: $M_0 = (\mu(p_1), \cdots, \mu(p_m))$ where $m$ is the number of places.

*Behaviour of a P-net.* A P-net evolves by firing some *enabled* transitions. A transition is *enabled* if all its input places contain at least so many tokens as is the weight of the arcs from the place to the transition. An enabled transition may be fired and enable all the actions in the output places of the transition. There is a non-deterministic choice between the enabled transitions. Firing a transition modifies the markings of both input and output places. This may enable or disable other transitions. All enabled transitions may be fired. Therefore the evolution of the net describes a *marking net* which can be infinite. When a transition is fired, one token is removed from every input place of the transition and one token is added to every output place of the transitions. This is generalized by removing (resp. adding) the number of tokens corresponding to the weight of the arcs from the input place to the transition (resp. to the weight of the arcs from the transition to the output place).

### 2.2 An Overview of B Abstract Systems

An *abstract system* [1,3] describes a mathematical model of a system behaviour[1]. It is mainly made of a state description (constants, variables and invariant) and several *event* descriptions. While *abstract machines* are the basic structures of the earlier operation-driven approach of the B method, *abstract systems* are the basic structures of the so-called *event-driven* B, and replace abstract machines. Abstract systems are comparable to Action Systems [4]; they describe a nondeterministic evolution of a system through guarded actions. Dynamic constraints can be expressed within abstract systems to specify various liveness properties [3,8]. The state of an abstract system is described by variables and constants linked by an invariant. Variables and constants represent the data space of the system being formalized. Abstract systems may be refined like abstract machines [8,2].

**Data of an Abstract System** At a higher level an abstract system models and contains the data of an entire model, be it distributed or not. Abstract systems have been used to formalize the behaviour of various (including distributed) systems [1,7,8,2]. Considering a global vision, the data that are formalized within the abstract system may correspond to all the elements of the distributed system.

[1] A system behaviour is the set of its possible transitions from state to state beginning from an initial state.

**Events of an Abstract System** Within B, an event is considered (like in the approach of Action Systems) as the observation of a system transition. Events are spontaneous and show the way a system evolves. An event has a *guard* and an *action*. It may occur or may be observed only when its guard holds. The action of an event describes with generalized substitutions how the system state evolves when this event occurs. Several events can have their guards held simultaneously; in this case, only one of them occurs. The system makes internally a nondeterministic choice. If no guard is true the abstract system is blocking (deadlock). An event has one of the general forms (Fig. 1) where

```
name $\widehat{=}$ /* event name */
    SELECT
        $P_{(gcv)}$
    THEN
        $GS_{(gcv)}$
    END
```

(SELECT Form)

```
name $\widehat{=}$ /* event name */
    ANY $bv$ WHERE
        $P_{(bv,gcv)}$
    THEN
        $GS_{(bv,gcv)}$
    END
```

(ANY Form)

**Fig. 1.** General Forms of Events

$gcv$ denotes the global constants and variables of the abstract system containing the event; $bv$ denotes the bound variables (variables bound to ANY). $P_{(bv,gcv)}$ denotes a predicate $P$ expressed with the variables $bv$ and $gcv$; in the same way $GS_{(bv,gcv)}$ is a generalized substitution $S$ which models the event action using the variables $bv$ and $gcv$. The SELECT form is just a particular case of the ANY form. The guard of an event with the SELECT form is $P_{(gcv)}$. The guard of an event with the ANY form is $\exists(bv).P_{(bv,gcv)}$.

**Semantics and Consistency.** An abstract system describes a mathematical model that simulates the behaviour of a system. Its semantics is based on the invariant and is guaranteed by proof obligations (POs). The *consistency* of the model is established by such proof obligations: *i) the initialisation should establish the invariant*; *ii) each event of the given abstract system should preserve the invariant of the model* (one must prove these POs). The proof obligation of an event with the ANY form is:

$$I_{(gcv)} \wedge P_{(bv,gcv)} \wedge \mathsf{term}(GS_{(bv,gcv)}) \Rightarrow [GS_{(bv,gcv)}]I_{(gcv)}$$

where $I_{(gcv)}$ stands for the invariant of the abstract system. The predicate $\mathsf{term}(S)$ expresses that the substitution $S$ terminates.

# 3 Embedding Petri Nets into Event-B

## 3.1 Embedding techniques

Embedding techniques are introduced in [5] and provide a methodology to reuse existing logical frameworks for formal analysis. Embedding techniques are intensively used for method integration and mechanization of notations [6,10,16]. There are two main embedding techniques: *shallow embedding* and *semantic embedding* (also called *deep embedding*). The first technique deals with the translation of specifications (objects of a formalism) to semantically equivalent objects in the target formalism. Nevertheless, the mapping from the language constructs to their semantic representations is part of the metalanguage (support of the source language). In the case of semantic embedding, the complete semantics of a source formalism is translated into the target formalism: both syntax and semantics of the source language are formalized inside the target language logic. That means, the mapping from language constructs to their semantic representations is part of the target language logic. Consequently, using semantic embedding, we do not need only the (semantic preserving) syntactic translation of the constructs but also the semantics to be translated into the target logic. The choice of one of the techniques depends on the envisaged goal.

### 3.2 Embedding the Structure of Petri Nets within B

Embedding the structure of a P-net into B (Fig. 2) consists in describing the graph associated to the P-net. The 4-tuple which describes a net $N$ is encoded with the set of places $(places)$, the set of transitions $(transitions)$, and the two relations between places and transitions $(placesBefore, placesAfter)$. Additionally we have the marking functions for the places: $mu$. We also consider the weights of the arcs; they are natural number greater or equal to the unit. The input arc weights are described by the function $weightBefore$. The output arc weights are described by the function $weightAfter$. Therefore some invariant properties may be added. This results in an event-less B abstract system (Fig. 2) which captures only the graph structure of a marked net $(N, mu)$. It remains to deal with the behavioural semantics of the net. This is based on the marking of the net and its transitions.

### 3.3 Embedding Petri Nets Evolution Semantics into B

A P-net evolves by firing the enabled transitions. From a given marking, firing one of the enabled transitions, leads to a new marking of the net and so on. This is embedded in event-B by an abstract system whose events correspond to the transition firing.

A P-net transition may be formalized (at first approximation) as a B event (see Fig. 3) with a guard which expresses that all the input places of the transition have the required number of tokens and a body (a generalized substitution) which expresses the update of input places (by removing the necessary tokens) and the update of output places (by adding the appropriate number of tokens). B events are instantaneous and their effect can cause the occurrence of other events. This copes well with the semantics of P-net: the firing of a transition $t_i$ is instantaneous and thus can lead to the firing of other transitions which have the output places of $t_i$ among their input places.

**Basic Petri net** Here *basic* Petri net means that actions (data+operations) are not attached to the places nor to the transitions. The arc weight may be greater or equal to the unit. The guard for firing a transition $t_i$ is that all its input places $pp$ have the required

SYSTEM *PetriNet*
SETS
  PLACE;  TRANSITION
VARIABLES
  $places, transitions, placesBefore, placesAfter, weightBefore, weightAfter, mu$
INVARIANT
  $places \subseteq$ PLACE
$\wedge$ $transitions \subseteq$ TRANSITION
$\wedge$ $placesBefore \in transitions \leftrightarrow places$  /* $placesBefore^{-1} = \Gamma_p$ */
$\wedge$ $placesAfter \in transitions \leftrightarrow places$  /* $placesAfter = \Gamma_t$ */
$\wedge$ $placesBefore = \mathrm{dom}(weightBefore)$
$\wedge$ $placesAfter = \mathrm{dom}(weightAfter)$
$\wedge$ $weightBefore \in transitions \times places \mathrel{+\!\!\!\!\to} \mathbb{N}$
$\wedge$ $\mathrm{dom}(weightBefore) = placesBefore$
$\wedge$ $weightAfter \in transitions \times places \mathrel{+\!\!\!\!\to} \mathbb{N}$
$\wedge$ $\mathrm{dom}(weightAfter) = placesAfter$
$\wedge$ $mu : places \rightarrow \mathbb{N}$

**Fig. 2.** A Partial B system encoding a P-net

number of tokens:

$$\begin{array}{l} ti \in transitions \wedge \\ \forall\, pp.(pp \in placesBefore[\{ti\}]) \Rightarrow \mu(pp) \geqslant weightBefore(ti, pp) \end{array}$$

The basic effect of firing a transition is the update, via the $\mu$ function, of the input and output places according to the input and output arcs. Let $pbef = placesBefore[\{ti\}]$ be the inout places of $ti$ and $paft = placesAfter[\{ti\}]$ the output places of $ti$. The update of the places after the transitions is:

$$\begin{array}{l} \mu := \mu \mathrel{<\!\!+} \{pp, vv \mid pp \in pbef \wedge vv = mu(pp) - weightBefore(ti, pp)\} \\ \qquad \mathrel{<\!\!+} \{pp, uu \mid pp \in paft \wedge uu = mu(pp) + weightAfter(ti, pp)\} \end{array}$$

Note that in the P-nets some places may be both input place and output place; those place need a cumulative update. Therefore we have a more general update performed as follows; let $pbef = placesBefore[\{ti\}] - placesAfter[\{ti\}]$ be the places at input only,
$paft = placesAfter[\{ti\}] - placesBefore[\{ti\}]$ be the places at output only and
$pcom = placesBefore[\{ti\}] \cap placesAfter[\{ti\}]$ the places being in input and output.

The update of $\mu$ is rigorously captured by:

$$\begin{array}{l} \mu := \mu \mathrel{<\!\!+} \{pp, vv \mid pp \in pbef \wedge vv = mu(pp) - weightBefore(ti, pp)\} \\ \qquad\qquad \mathrel{<\!\!+} \{pp, uu \mid pp \in paft \wedge uu = mu(pp) + weightAfter(ti, pp)\} \\ \qquad\qquad \mathrel{<\!\!+} \{pp, mm \mid pp \in pcom \wedge mm = mu(pp) - weightBefore(ti, pp) \\ \qquad\qquad\qquad\qquad + weightAfter(ti, pp)\} \end{array}$$

Therefrom, the firing of a transition $t_i$ is translated with a single B event **`event_tr`** (Fig. 3) which works for every transition $t_i$ in a non-deterministic way. The variables $mupbef$ and $mupaft$ model with B generalized substitutions the update of the $\mu$ function as described above. The notation $rel_1 \mathbin{<\!\!+} rel_2$ denotes the overriding of a relation by another one.

```
event_tr ≙            /* firing of any transition t_i */
 ANY $t_i$ WHERE
    $t_i \in transitions \wedge \forall pp.(pp \in placesBefore[\{ti\}] \Rightarrow \mu(pp) \geqslant weightBefore(t_i, pp))$
 THEN
    LET $pbef, paft, pcom$ BE
    $pbef = placesBefore[\{t_i\}] - placesAfter[\{t_i\}]$
    $\wedge\ paft = placesAfter[\{t_i\}] - placesBefore[\{t_i\}]$
    $\wedge\ pcom = placesAfter[\{t_i\}] \cap placesBefore[\{t_i\}]$
    IN
    /* update of places after t_i */
    $mu := mu \lhd\!\!\!+ \{pp, vv \mid$
          $pp \in pbef \wedge vv \in NAT \wedge vv = mu(pp) - weightBefore(ti, pp)\}$
       $\lhd\!\!\!+ \{pp, uu \mid pp \in paft \wedge uu \in NAT \wedge uu = mu(pp) + weightAfter(ti, pp)\}$
       $\lhd\!\!\!+ \{pp, mm \mid pp \in pcom \wedge mm \in NAT \wedge mm = mu(pp) - weightBefore(ti, pp)$
            $+weightAfter(ti, pp)\}$
    END
END
```

**Fig. 3.** A shape of a B event capturing the evolution of a basic P-net

We captured the behavioral semantics of basic P-nets with a B abstract system with a *single event* representing the transitions of the net. This abstract system simulates the evolution of the P-net. Using a single event for all transitions instead of one event per transition simplifies the generalisation and the reasoning on the embedding; indeed only the structure of a parameter P-net needs to be translated for each new project.

**Generic Structure of the Embedding** We show in Figure 4 the B generic structure which holds all P-net model; it is the abstract system named $EmbeddedPN$. We separate the encoding of the semantics ($EmbeddedPN$) which works for any P-net and the static structure part ($PetriNet$) which is specific to a problem and should be included for a given problem. The static part (in the $PetriNet$ abstract system) is completed with some variables: $pl\_actions$ is the set of actions attached to the places. The injective (total) function[2] $pl\_treatment \in places \rightarrowtail pl\_actions$ records the action located in each place; a specific element $nullaction$ is used for the initialisation and for actionless places. Technically, the use of the $nullaction$ avoids a blocking of the system at the initialisation, were all the actions should be disabled (i.e. their guards are false).
The system $EmbeddedPN$ has two variables: the relation $trans\_places$ records, for the

[2] It is injective because we need the reverse function.

currently fired transition(s), the output places which are not yet processed; the function $guard\_P\_actions$ is used to get the guard of each place action.
The single event **`event_tr`** manages the firing of transition and thus the evolution of the considered net. This event is improved and is replaced in the following sections by two (or several events according to the considered policy) related events (`action_ak, fire_transition`).

```
SYSTEM EmbeddedPN
INCLUDES
   PetriNet      /* any described P-net; this is a parameter */
VARIABLES
   $guard\_P\_actions, trans\_places$
INVARIANT
   $guard\_P\_actions \in pl\_actions \rightarrow BOOL$
$\wedge$  $trans\_places \in transitions \leftrightarrow places$
INITIALISATION
   $guard\_P\_actions := ((pl\_actions - \{nullaction\}) \times \{FALSE\})$
            $\cup\{(nullaction, TRUE)\}$
$||$  $trans\_places := \{\}$
EVENTS
   event_tr $\widehat{=}$ $\cdots$    /* for any transition ti */
END
```

**Fig. 4.** Generic Structure of the Embedding

Therefrom we extend the embedding to cover more complicated cases: action management. Indeed, according to their types (place/transitions, conditions/event, resources, etc), P-nets may deal with data and actions (or treatments) in various manners.

In some P-nets the places with tokens may model availability of data; in this case an action may be associated to the transitions related to the places.

In other models, some places may contain action which is then guarded by one or several transitions. It is for instance the case in a net modelling a process writing some data in exclusion with other writer processes; a specific place is often used in such a case in order to handle the exclusion between processes. Therefore there is not a single way to embed the P-nets. We investigated both cases of action (or treatment) attachments: attachment of treatments to the places and to the transitions.

### 3.4 Treatement of Non-Basic Nets

In the previous section, we considered the evolution of basic P-nets; no specific policies or treatments are considered.

**High Level Petri Nets** High Level Petri Nets (HLPN) were introduced to overcome the problem of the explosion of the number of elements needed for large computer systems. HLPN use *i)* structured data to model the tokens, and algebraic expressions to annotate the net elements; *ii)* transition modes to describe more elaborated operations/actions. Within HLPN the enabling of a transition depends not only on the availability of the tokens but also on their nature. There are several achievements of HLPN [13]; Predicate/Transition-Nets [9] and Colored Petri Nets [12,15] are two forms of HLPN. In this article, we consider an abstraction of the ideas of HLPN. Actions (treatments or operations) may be associated to places and transitions of the nets. This corresponds to the idea of structured tokens, typed places and typed transitions, and more generally the execution of some operations associated to the places or to the transitions of a net. Accordingly, we propose a generic treatment of the whole.

The study is achieved step by step; first we examine the formalisation in the case where actions are attached to places only. Then we study the cases where they are associated to transitions. Finally we consider the general case where actions are attached to both the places and the transitions.

**Embedding into B of Petri Nets with Actions Attached to Places** The action attached to a place should be achieved when the input transition associated to the place (the guard) is fired. Thereby *each action in a place of a P-net is translated as a (guarded) event of the B abstract system*.

In practice, actions need some time to be completed. Therefore firing a transition may be achieved in two steps: *i)* enabling the guard of all the actions attached to the output places of the transition; *ii)* launching non-deterministically these *involved actions*. All of them should be performed in any order.

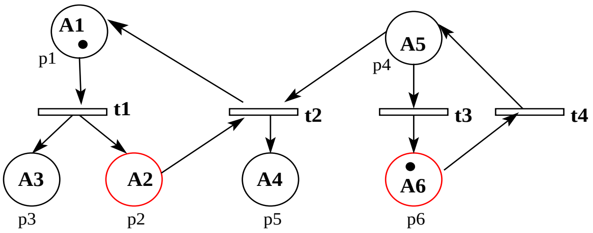

**Fig. 5.** Interdependent Actions

This raises some questions: what should be the duration of the actions and the enabling of other transitions? Should we wait for the completion of an action before considering another action? What is the scheduling of the enabled transitions and enabled actions? Considering these questions with respect to the Figure 5, one has an idea of the complexity of the scheduling of actions; the transition t1 enables the actions {A2, A3}; t2 enables the actions {A4, A1}; t3 enables the action {A6}; t4 enables the action {A5}.

These actions are interdependent because the places that contain them are either an input place or an output place of the fired transitions. There are cycles; for example, firing repeatedly the transitions t3 and t4.

To deal with the current situation, we use the previously defined (see Section 3.3) variables $pl\_treatment$, $pl\_actions$ and $guard\_P\_actions$. The firing of a transition $ti$ is handled with two events which correspond to the two steps distinguished above.

**Step. 1** The first step of the transition firing is captured with the B event **fire_transition_tr** given in Figure 6. The output places of a transition $t_i$ are specified as: $paft = placesAfter[\{ti\}]$. The involved actions associated to these places are: $involved\_actions = paft \lhd pl\_treatment$.

The guards of the involved actions attached to the output places of the fired transition are enabled ($\forall\ Ai\ \in\ placesAfter[\{ti\}].\, guard(Ai)\ :=\ TRUE$). The function $guard\_P\_actions$ is updated in order to enable the guards. This is done with a Cartesian product: $ran(involved\_actions) \times \{TRUE\}$. The marking of the input places is updated. The fired transition and its output places are recorded in the relation $trans\_places$; this is necessary for the scheduling of involved actions. Indeed *all the actions of the output places should be performed before the actions of the possible transitions they can enable.*

```
fire_transition_tr $\widehat{=}$            /* for any transition ti */
    ANY $t_i$ WHERE
        $t_i \in transitions$
    $\wedge\ \forall pp.(pp \in placesBefore[\{ti\}] \Rightarrow \mu(pp) \geqslant weightBefore(t_i, pp))$
    THEN
        LET $pbef,\ paft,\ involved\_actions$ BE
            $pbef = placesBefore[\{t_i\}] \wedge paft = placesAfter[\{t_i\}]$
        $\wedge\ involved\_actions = paft \lhd pl\_treatment$
        IN
            /* enabling the guards of involved actions */
        $guard\_P\_actions := ran(involved\_actions) \times \{TRUE\}$
            /* udpate of input places of $t_i$, */
            /* output places of $t_i$ will be updated after the actions */
    ||  $mu := mu \mathbin{<\!\!+} \{pp, vv \mid pp \in pbef \wedge vv \in NAT \wedge vv = mu(pp)$
                $-weightBefore(ti, pp)\}$
              /* update of places to be treated after the fired transition */
    ||  $trans\_places := trans\_places \cup (\{t_i\} \times paft)$
        END
END
```

**Fig. 6.** Piece of the dynamic part of the generic structure (a)

Since the B events are atomic we cannot update the marking of output places during the first step; they will eventually enable other transitions which will take place. Moreover, to cope with practical application of P-nets, one has to consider the "run until completion" of the various actions during their scheduling.

**Step. 2** The second step of the firing is captured with the event **`action_Ak`** (see Fig. 7). One B event is described for each action associated to a place. This enables us to handle the high level aspect of the net; indeed the treatments depend on the tokens and on the transitions. The guard of each action is maintained (to TRUE) until the action is started and performed. The actions attached to the output places which are still enabled, are non-deterministically performed; they are recorded in (the range of) $trans\_places$. But, the actions in the places contained in $trans\_places$ can be performed at any time (due to the non-determinism of event occurrence). When an action is completed its guard is disabled and the number of tokens of the related place is updated: the function $trans\_places$ is updated, the $mu$ function is updated to set the marking of output places.

```
action_Ak $\widehat{=}$          /* for an action Ak (attached to a place pp) */
    ANY $Ak$ WHERE
        $Ak \in actions$
        $\wedge\ guard\_P\_actions(Ak) = TRUE$     /* one of the enabled actions*/
    THEN
        LET $pp, tr, weiga, \cdots$ BE
        $\wedge\ \ pp = pl\_treatment^{-1}(Ak)$     /* the place associated to Ak */
        $\wedge\ \ tr = trans\_places^{-1}(pp)$     /* the transition before pp */
        $\wedge\ \ weiga = weightAfter(tr, pp)$     /* weight of the edge */
    $\wedge\ \ \cdots$        /* unused parts, cut */
        IN
        $guard\_P\_actions(Ak) := FALSE$
    $||\ \ mu(pp) := mu(pp) + weiga$
    $||\ \ trans\_places := trans\_places - \{(tr, pp)\}$
    $||\ \ \cdots$   /* location of an effective Ak */
        END
    END
```

**Fig. 7.** Piece of the dynamic part of the generic structure (b)

However, there are some shortcomings with the current situation. There is a kind of loss of priority between actions: if the effect of one of the currently enabled actions contributes to fire another transition, the actions which are enabled by this latter transition can be performed before the actions already enabled (this comes fatally from the substitution $trans\_places := trans\_places \cup (\{t_i\} \times paft)$).

Another shortcoming is the following: when there are cycles, an enabled guard (of an action) can be overwritten; that is, the enabling condition can be observed again whereas the already enabled action is not yet performed.

We solve these problems in the general case presented later on, by using priorities.

**Embedding into B of Petri Nets with Actions Attached to Transitions** In the same way as for the previous case with places, a total function $tr\_treatment \in transitions \rightarrow tr\_actions$ records the action associated to each transition.

$tr\_actions$ is used for the set of actions attached to all the transitions; it is defined in the static structure ($PetriNet$). When an enabled transition is fired, its associated action should be performed before the update of the marking of the output places, otherwise another transition may take the priority over the current one.

Several transitions may share the same input place(s). But, when the latter has the necessary number of tokens to enable the transitions which share the place, only one of the enabled transitions is fired. Therefore two steps are necessary to handle the firing of a transition. In a first step, one of the enabled transitions is non-deterministically selected; the guard of the action associated to this transition is enabled. The marking of all the input places is updated. This is quite similar to the event **fire_transition_tr**. In a second step, the transition action is performed; its guard is disabled and, the marking of the output places is updated. These places may enable other transitions and so forth. We get two B events corresponding to the described steps: *i)* a firing event which is used to select a transition and to update the input places; this event deals with all the enabled transitions; *ii)* each transition action has an event with its associated guard which depends on the marks of input places.

**Embedding into B of Petri Nets with Actions Attached to both Places and Transitions** In the current case, when a transition is fired, the attached action is enabled and the marking of the output places of the transition is updated; these output places have actions which should be enabled. After that, the transition action is performed, it enables the actions attached to the output places. Moreover, the actions linked to the places should be performed before enabling the transitions linked to them. In order to embed this semantics, we use two functions additionally with the preceding variables: $enabled\_P\_actions$ for the currently enabled place actions and $enabled\_T\_actions$ for the currently enabled transition actions. Remind that $trans\_places$ records which output places are not yet processed for the currently fired transition.

The embedding is achieved according to priority rules. The priority between actions are handled as follows. A transition is fired if *i)* the input places have the required number of tokens, *ii)* there is no previous enabled place action not yet performed; this is checked with $(trans\_places = \{\})$. Indeed when a transition is fired, its action is enabled and it enables some (output) place actions. These latter should be performed before firing another transition. This policy solves the problem of guard overwriting.

Therefrom the event **fire_transition_tr** is modified as described in Figure 8.

The remaining events (not detailed here) are the following:
**enable_transition_action_guard**; it sets the guard of an enabled transition action to $true$, then it disables the transition guard.
**enable_place_action_guard**; it sets the guard of an enabled place action to $true$, updates the $mu$ function and updates $trans\_places$ by removing the treated place;
**launch_transition_action_aj**; it launches one of the transition action whose guard is $true$ and then it sets the guard to $false$;
**launch_place_action_ak**; this one launches a place action whose guard is enabled, then the guard is disabled.

All these five events (of the abstract system $EmbeddedPN$) simulate an interleaving run of the firing of transition actions and place actions, but priority is employed to avoid wrong behaviour of the actions. The entire system is checked for consistency using Atelier B and analysis issues are experimented with various case (small-size) studies.

```
fire_transition_tr $\widehat{=}$            /* for any transition ti */
    ANY $t_i$ WHERE
        $t_i \in transitions \wedge \forall pp.(pp \in placesBefore[\{ti\}] \Rightarrow \mu(pp) \geqslant weightBefore(t_i, pp))$
    $\wedge$ trans_places = {} $\wedge$ (enabled_P_actions $\triangleright$ { TRUE }) ={}
        /* and there is no action to be treated (this is priority handling) */
    THEN
        LET $pbef, paft, involved\_actions$ BE
            $pbef = placesBefore[\{t_i\}] \wedge paft = placesAfter[\{ti\}]$
        $\wedge$  $involved\_actions = paft \lhd pl\_treatment$
        $\wedge$  $\cdots$ /* unused here, cut */
        IN
            $enabled\_T\_actions(t_i) := TRUE$
             /* enable the action guards of the involved places */
        $||$  $enabled\_P\_actions := ran(involved\_actions) \times \{TRUE\}$
        $||$  $mu := mu \lhd\!\!\!+ \{pp, vv \mid pp \in pbef \wedge vv \in NAT \wedge vv = mu(pp)$
                      $-weightBefore(ti, pp)\}$
        $||$  $trans\_places := \{t_i\} \times paft$
        END
    END
```

**Fig. 8.** Piece of the dynamic part of a Petri Net with Place and Transition Actions

# 4 Analysis Issues

## 4.1 Analysis of Petri Nets

Very often, two classes of properties are studied on P-nets: one is about the boundedness of the nets. For example the accumulation of tokens in a place is symptomatic of a bad functioning of a model. The second class is about the liveness of the nets. By studying the reachability of certain marking, one can detect deadlock freedom for example. In all these cases, the marking graph (the set of reachable markings) should be computed. This aspect of the analysis may raise some problems. The size of the graph may be too large for an analysis in a reasonable time; the graph may also be infinite. When the graph is infinite, a covering graph is used instead; it enables to check a part of the desired properties.

Three main classes of analysis techniques [17,19] for P-nets are:
*Reachability analysis*: it is based on state space exploration/reduction techniques using model checking. The main idea is to construct an occurrence graph (a directed graph) which has a node for each reachable system state (a marking) and an edge for each possible state transition. The analysis is then based on such graph.
Reachability is like a simulation of the modelled system execution. It allows for a rapid analysis of the system to check for its functionalities.
*Structural analysis*: algebraic analysis are applied here.
*Invariant analysis*: it consists to check that some properties associated to the places are satisfied for all reachable states (a net marking) of the modelled system.

The advantages of the first analysis techniques are: a graph is constructed and analysed systematically; the constructed graph may be very large; but there are techniques which work with minimized graphs. The main disadvantage is that, such a graph may become very large, even for very small systems, making the analysis unpractical due to state explosion problem.

One of the aspects on which this work contributes in is the definition of the basis for the combined use of analysis techniques and tools. The available B platforms may be used to analyze the safety properties of systems which are modelled with P-nets.

### 4.2 An Illustration: Producer-Consumer with Semaphore

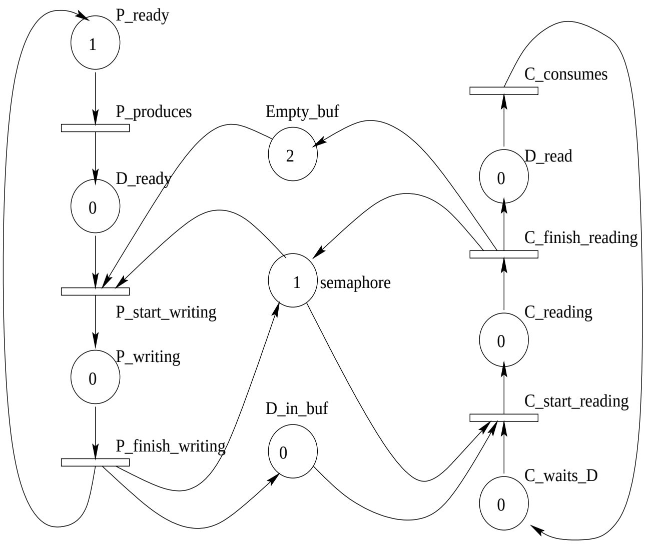

**Fig. 9.** A producer-consumer example

We described and checked the producer-consumer system depicted in Figure 9 using our approach. Only the description of the abstract system $PetriNet$ is given, and here it encodes the specific Producer-Consumer net. This encoding of the Petri Net is then included in the system $EmbeddedPN$ which is not changed (it already gathers all P-nets semantics). The specifications are given in the appendix.

Additionally to the properties that may be analysed in a standard Petri net platform, some safety properties that may be analysed using the B tools are:

- Boundedness of some places: the places Empty_buf and D_in_buf (see Fig. 9) are bounded. This is formalized as the following predicate which is added to the invariant:
  $mu(Empty\_buf) \leqslant 2 \wedge mu(D\_in\_buf) \leqslant 2$ (prop1)
- There is not a wrong usage of the resources (here the buffer):
  $mu(Empty\_buf) + mu(D\_in\_buf) = 2$ (prop2)
- The system is *live*; that means there is always at least one transition which can be fired; this is formalized with:
  $placesBefore^{\sim}[dom(nmu \rhd \{ii \mid ii \in \mathbb{N} \wedge ii > 0\})] \neq \{\}$

The properties described above are integrated in the invariant of the our B specification (see appendix A) of the producer-consumer and proved.This illustrates how we may manage the modelling and analysis work through Petri nets and B.

## 5 Conclusion and Further Work

We presented an embedding of Petri nets formalisms into the B abstract system formalism. The embedding is systematic and it covers basic P-nets as well as high level nets. The current work fills a gap between the widely practiced P-nets formalism and the proof-based development technologies especially the B method which is based on abstract machines, refinement and theorem proving. This is a step towards a multi-facet analysis framework for relating discrete system modelling techniques.
*Results.* We have provided a two-level embedding infrastructure made of a generic B abstract system that may be used to describe any Petri net and, an abstract system that includes (genericity) the first one and whose events capture the semantics of Petri nets evolution. Various policies concerning high level P-nets have been considered. Concretely we may combine the use of P-nets and B method in the same project; for example we may begin the modelling with an existing graphical tool dedicated to the P-nets and then follow with the B method for the related aspects. This work is generally related to works on embedding techniques but it is specifically related to the work by Sekerinski and Zurob [20] on Statecharts and B. In this work, unlike our approach, the abstract structure of Statecharts is translated into the semantically equivalent one in B. In or work the the translation is performed by considering the global semantics instead.
*Further work.* Ongoing effort focuses on the automation of all the chains from a P-Net tool to the B tools. We investigated the transformation of the XML outputs of the tools such as the PEP tool [3] into a B machine (see appendix **??**). The result is to be passed as the included machine. But, many experiments of various size are still needed for the scalability of our translation process. Meanwhile, user-friendly tools to facilitate the combination of the techniques are to be developed.

## References

1. J-R. Abrial. Extending B without Changing it (for developping distributed systems). *Proc. of the 1st Conf. on the B method, H. Habrias (editor), France*, pages 169–190, 1996.
2. J-R. Abrial, D. Cansell, and D. Mery. Formal Derivation of Spanning Trees Algorithms. In D. Bert et al., editor, *ZB'2003 – Formal Specification and Development in Z and B*, volume 2651 of *LNCS*, pages 457–476. Springer-Verlag, 2003.
3. J-R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In *Proc. of the 2nd Conference on the B method, D. Bert (editor)*, volume 1393 of *Lecture Notes in Computer Science*, pages 83–128. Springer-Verlag, 1998.
4. R.J. Back and R. Kurki-Suonio. Decentralisation of Process Nets with Centralised Control. In *Proc. of the 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142, 1983.
5. R. Boulton, A. Gorgon, M.J.C. Gordon, J. Hebert, and J. van Tassel. Experience with Embedding Hardware Description Language in HOL. In *Proc. of the International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience*, pages 129–156, North-Holland, 1992. IFIP TC10/WG 10.2.

[3] `sourceforge.net/projects/peptool`

6. J. P. Bowen and M. J. C. Gordon. A Shallow Embedding of Z in HOL. *Information and Software Technology*, 37(5-6):269–276, 1995.
7. M. Butler and M. Walden. Distributed System Development in B. *Proc. of the 1st Conference on the B method, H. Habrias (editor), France*, pages 155–168, 1996.
8. D. Cansell, G. Gopalakrishnan, M. Jones, and D. Mery. Incremental Proof of the Producer/Consumer Property for the PCI Protocol. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *ZB'2002 – Formal Specification and Development in Z and B*, volume 2272 of *LNCS*, pages 22–41. Springer-Verlag, 2002.
9. H. J. Genrich. Predicate/Transition Nets. In W. Brauer, W. Reisig, and G. Rozenber, editors, *Petri Nets: Central Models and their Properties, Advances in Petri Nets(1986)*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer-Verlag, 1987.
10. A. W. Gravell and C. H. Pratten. Embedding a Formal Notation: Experiences of Automating the Embedding of Z in the Higher Order Logic of PVS and HOL. In *[11]*, pages 73–84, 1998.
11. J. Grundy and M. Newey, editor. *Supplementary Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics: Emerging Trends, (TPHOL'98)*. Australian National University, 1998.
12. K. Jensen. Coloured Petri Nets and the Invariant Method. *TCS*, 14:317–336, 1986.
13. K. Jensen. Coloured Petri Nets Vol. I-III. In *EATCS Monographs on Theoretical Computer Science*, EATCS. Springer-Verlag, 1992-1996.
14. L. M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad-hoc Networks. In *Proceedings of INT'04*, volume 3147 of *LNCS*, pages 248–269. Springer-Verlag, 2004.
15. Lars Michael Kristensen, Jens Bæk Jørgensen, and Kurt Jensen. Application of Coloured Petri Nets in System Development. volume 3098 of *LNCS*, pages 626–685. Springer-Verlag, Jan 2004.
16. C. Muñoz and J. Rushby. Structural Embeddings: Mechanization with Method. In J. Wing and J. Woodcock, editor, *Proc. of the World Congress in Formal Methods (FM99)*, volume 1708 of *Lecture Notes in Computer Science*, pages 452–471, France, 1999. Springer-Verlag.
17. T. Murata. Petri-Nets: Properties, Analysis and Applications. In *Proc. IEEE*, volume 77, pages 541–580. IEEE, 1989.
18. W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer, 1998.
19. W. Reisig and G. Rozenberg, editors. *Lectures on Petri nets I: Basic models and II: Applications*, volume 1491/1492 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
20. E. Sekerenski and R. Zurob. Translating Statecharts to B. In *Proc. of the Integrated Formal Methods (IFM'2002)*, volume 2335 of *Lecture Notes in Computer Science*, UK, May 2002. Springer-Verlag.

# A The B machine of the Petri Net

```
/∗ Encoding of the Producer−Consumer Petri−Net ∗/
MACHINE
        ProdCons
SETS
/∗−−−− fill in the two sets PLACE and TRANSITION −−−−−−∗/
        PLACE = {P_ready, D_ready, P_writing, Empty_buf, semaphore,
        D_in_buf,D_read, C_reading, C_wait_D}
;       TRANSITION = {P_produces, P_start_writing, P_finish_writing ,
        C_consumes, C_finish_reading , C_start_reading }
/∗−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−∗/
;       ACTION = {aj, ak, tai , taj , nullaction } /∗ actions associated ∗/
;       NET_Type = {pure, unspecified , colored}
;       NET_Mode = {edition, analysis}  /∗ edition ∨ analysis mode ∗/
VARIABLES
        net_type            /∗ PN type                ∗/
,       net_mode            /∗ PN mode                ∗/

CONSTANTS /∗ parameter of the machine ∗/
        places              /∗ the places in the PN   ∗/
,       transitions         /∗ the transitions in the PN ∗/
,       placesBefore        /∗ places before a transition ∗/
,       placesAfter         /∗ places after a transition  ∗/
,       weightBefore        /∗ weight of an edge before a transition ∗/
,       weightAfter         /∗ weight of an edge after a transition ∗/
,       pl_actions          /∗ all actions attached to the places ∗/
,       tr_actions          /∗ all actions attached to the transitions ∗/
,       pl_treatment        /∗ treatment (or actions) associated to each place ∗/
,       tr_treatment        /∗ treatment (or actions) associated to each transition ∗/
,       mu                  /∗ marking of each place  ∗/

PROPERTIES
        places          ⊆ PLACE
∧       transitions     ⊆ TRANSITION
∧       placesBefore    ∈ transitions ↔ places
∧       placesAfter     ∈ transitions ↔ places
∧       weightBefore    ∈ transitions × places ⇸ NAT
∧       dom(weightBefore) = placesBefore
∧       weightAfter     ∈ transitions × places ⇸ NAT
∧       dom(weightAfter) ⊆ placesAfter
∧       dom(placesBefore) ⊆ transitions
              /∗      every transition has at least one place after it ∗/
∧       dom(placesAfter) ⊆ transitions
/∗−−−−− Fill in description of the Petri −Net to be studied ∗/
∧       places = { P_ready, D_ready, P_writing , Empty_buf, semaphore, D_in_buf,
D_read, C_reading, C_wait_D   }
∧       transitions = {P_produces, P_start_writing , P_finish_writing ,
C_consumes, C_finish_reading , C_start_reading }
```

```
/∗−−−−− Fill in, using maplet ∗/
∧          placesBefore  = {
P_produces ↦ P_ready,
  P_start_writing  ↦P_ready,
  P_start_writing  ↦Empty_buf,
  P_start_writing  ↦semaphore,
  P_finish_writing ↦P_writing,
 C_consumes ↦D_read,
 C_finish_reading  ↦C_reading,
 C_start_reading  ↦C_wait_D,
 C_start_reading  ↦D_in_buf,
 C_start_reading  ↦semaphore }
∧          placesAfter  = {
P_produces ↦ D_ready,
  P_start_writing ↦ P_writing,
  P_finish_writing ↦P_ready,
  P_finish_writing ↦D_in_buf,
  P_finish_writing ↦semaphore,
 C_consumes ↦C_wait_D,
 C_finish_reading ↦ D_read,
 C_start_reading  ↦C_reading }
/∗−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−∗/
∧          weightAfter  = placesAfter∗{1} /∗ weight of edge after a transition  ∗/
∧          weightBefore  = placesBefore∗{1}
∧          pl_actions  ⊆ ACTION
                    /∗  the actions controlled by the PNet ∗/
∧          tr_actions  ⊆ ACTION
                    /∗  all the transition actions controlled by the PN ∗/
∧          nullaction  ∈ pl_actions  ∩  tr_actions
∧          pl_treatment  ∈ places  ⇸  pl_actions  /∗ which place has what action  ∗/
∧          tr_treatment  ∈  transitions  ⇸  tr_actions  /∗ which transitions has what action  ∗/
∧          pl_actions  = { nullaction } /∗ actions sur les places ∗/
∧          tr_actions  =  { nullaction } /∗ actions sur les transitions  ∗/
∧          pl_treatment  = places∗pl_actions     /∗ ∗/
∧          tr_treatment  =  transitions ∗ tr_actions  /∗ ∗/
∧          mu        ∈ places  →       NAT
/∗−−−−−−Fill in the marking ∗/
∧ mu = {
 (P_ready ↦1), (Empty_buf↦2), (semaphore↦ 1), (D_ready↦0),
(P_writing↦0), (D_in_buf↦0), (D_read↦0), (C_reading↦0), (C_wait_D↦ 0)}
/∗−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−∗/
DEFINITIONS
          GUARD ≙ 𝔹
INVARIANT
          net_type  ∈        NET_Type
                    /∗        no cycle ∈ place_i ↪ trans−i ↪ place_i  ∗/
∧          (( net_type  = pure)  ⇒  ( placesBefore  ∩  placesAfter  = ∅))
∧          net_mode ∈        NET_Mode /∗ in analysis mode after the edition  ∗/
/∗−−−− Include here desired safety properties     ∗/
/∗ Shape ∈ (mode = analysis)  ≙> the properties  ∗/
```

```
/∗
∧ ((mu(Empty_buf) ⩽ 2) ∧ (mu(D_in_buf) ⩽ 2))        /∗ (prop1) ∗/
∧ (mu(Empty_buf) + mu(D_in_buf) = 2)                 /∗ (prop2) ∗/
/∗−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−∗/
INITIALISATION
        net_type := unspecified
||      net_mode := analysis
OPERATIONS
        set_edition_mode = /∗ set the mode to edition ∗/
        SELECT
                net_mode = analysis
        THEN
                net_mode := edition
        END
;
res ⟵ which_mode = /∗ what is the current mode ∗/
        BEGIN
                res := net_mode
        END
END
```

## B The B Structure of the Embedding Machine

```
/∗ Globale Machine Embeddeding A Petri Encoding ∗/
MACHINE
        EmbeddedPN_PT
INCLUDES
        ProdCons /∗ Parameter ; the embedded Machine ∗/
VARIABLES
        enabled_P_actions       /∗ currently enabled place actions ∗/
,       enabled_T_actions       /∗ currently enabled transition actions ∗/
,       guard_P_actions  /∗ for all actions ∗/
,       guard_T_actions
,       trans_places     /∗ which transition currently activates some places ∗/
,       nmu              /∗ a replacement of mu so as to update mu (¬ modifiable) easily ∗/

DEFINITIONS
    GUARD ≙ 𝔹

INVARIANT
    enabled_P_actions ∈ pl_actions ⇸ GUARD
        /∗ the currently enabled place actions ∗/
∧   enabled_T_actions ∈ tr_actions ⇸ GUARD
        /∗ the currently enabled transition actions ∗/
∧   guard_P_actions ∈ pl_actions → GUARD /∗ the guard of each place action ∗/
∧   guard_T_actions ∈ tr_actions → GUARD /∗ the guard of each transition action ∗/
. . .

END
```

## C The Experimental Toolchain Architecture

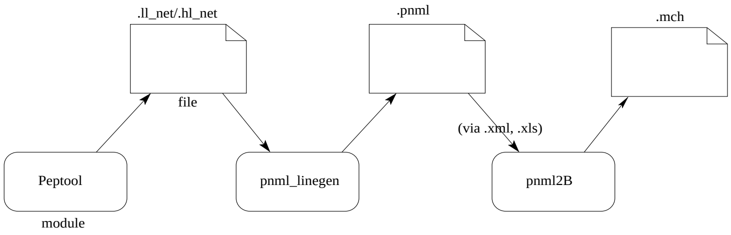

**Fig. 10.** Overview of the Experimental Toolchain

We have undertaken the development of a toolchain to mechanize our bridging process. Currently the encoding of a working Petri net is systematic but still manual; for this purpose the generic machine (PetriNet) is used as a pattern; only a few part of it is filled; that is the values of the sets PLACE and TRANSITION in the SETS clause and, the values of the variables places, transitions and last the variables placesBefore, placesAfter. In addition, we also have a pattern to introduce the desired properties.

The module pnml2b in the architecture (Fig. 10) is devoted to the mechanisation of this process.