Asynchronous links in the PBC and M-nets

Hanna Klaudel, Franck Pommereau

Proc. of ASIAN'99, LNCS 1742, pages 190-200, Springer, 1999. © 1999 Springer

Asynchronous links in the PBC and M-nets

Hanna Klaudel and Franck Pommereau

Université Paris XII, 61 Avenue du Général de Gaulle
94 010 Créteil, France
{pommereau,klaudel}@univ-paris12.fr

Abstract. This paper aims at introducing an extension of M-nets, a fully compositional class of high-level Petri nets, and of its low-level counter part, Petri Boxes Calculus (PBC). We introduce a new operator with nice algebraic properties which allows to express asynchronous communications in a simple and flexible way. With this extension, asynchronous communications become at least as simple to express as (existing) synchronous ones. Finally, we show how this extension can be used in order to specify systems with timing constraints.

Keywords: Petri Net, Petri Box Calculus, M-Net, Semantics, Timed Specification.

1 Introduction

M-nets, constructed at the top of the algebra of Petri boxes [4,3,13], are a fruitful class of high-level Petri nets which nicely unfold into low-level nets and thus allow to represent large (possibly infinite) systems in a clear and compact way. They are widely used now as a semantic domain for concurrent system specifications, programming languages, or protocols, *cf.* [6,7,11,1,14,2,12]. The most original aspect of M-nets with respect to other high-level net classes is their full compositionality, thanks to their interfaces, and a set of various net operations defined for them. Their interest is augmented by the ability to use in practice an associated tool, PEP [5], which also offers various implemented verification and analysis methods.

This paper defines a possibility to express asynchronous communication links at the M-net and PBC algebras level by introducing a new operator (**tie**). This extension mainly concerns the labelling of transitions, and now, in addition to usual synchronous communications (with the synchronisation operator **sy**), transitions may also export or import data through asynchronous links. It turns out that the **tie** operator has nice algebraic properties: idempotence, commutativity with itself and with the synchronisation, and also coherence (*i.e.*, commutativity) with respect to the unfolding. These properties allow to preserve the full compositionnality of the model.

As an application, we present a modeling of discrete time constraints in the M-nets. This allows to specify time-constrained systems and to analyse their unfoldings with existing tools (*e.g.*, PEP). We use a high-level featured clock which handles *counting requests* for the rest of the system. Asynchronous links are used to perform the necessary communications between the actions related to each request.

The next three sections briefly introduce M-nets and PBC, including the basis for our extension. Then, sections 5 and 6 give the definitions and the algebraic properties of the **tie** operation. Section 7 is devoted to the application of asynchronous links to discrete time modelling.

2 Basic Definitions

Let E be a set. A *multiset* over E is a function $\mu : E \to \mathbb{N}$; μ is finite if $\{e \in E \mid \mu(e) > 0\}$ is finite. We denote by $\mathcal{M}_f(E)$ the set of finite multi-sets over E, by $\oplus$ and $\ominus$ the sum and difference of multi-sets, respectively. $\odot$ is used to relate an element of E to the number of its occurences in a multi-set over E; in particular, a multi-set μ over E may be written as $\bigoplus_{e \in E} \mu(e) \odot e$, or in extended set notation, *e.g.*, $\{a, a, b\}$ for $\mu(a) = 2$, $\mu(b) = 1$ and $\mu(e) = 0$ for all $e \in E \setminus \{a, b\}$. We may also write $e \in \mu$ for $\mu(e) > 0$.

Let *Val* and *Var* be fixed but suitably large disjoint sets of *values* and *variables*, respectively. The set of all well-formed *predicates* built from the sets *Val*, *Var* and a suitable set of operators is denoted by Pr.

We assume the existence of fixed disjoint sets A_h of *high-level action symbols* (for transition synchronous communications) and B of *tie symbols* (for transition asynchronous links). We assume that each element $A \in \mathsf{A}_h$ has an arity $ar(A)$, and that there exists a bijection $\widehat{\ }$ on A_h (called *conjugation*), with $\widehat{\widehat{A}} = A$, $\widehat{A} \neq A$ and $ar(A) = ar(\widehat{A})$. We also assume that each element $b \in \mathsf{B}$ has a type $type(b) \subseteq Val$.

A *high-level action* is a construct $A(a_1, \ldots, a_{ar(A)})$ where $A \in \mathsf{A}_h$ (notice that we could have $\widehat{A}$ instead of A) and $a_i \in Val \cup Var$ ($1 \leq i \leq ar(A)$). A typical high-level action is, for instance, $A(a_1, a_2, 5)$ where $A \in \mathsf{A}_h$ and a_1, a_2 are variables. The set of all high-level actions is denoted by AX_h.

Similarly, a *low-level action* is a construct $A(v_1, \ldots, v_n) \in \mathsf{AX}_h$, where $v_i \in Val$ for all $i \in \{1, \ldots, n\}$. The set of all low-level actions is denoted by

A_l. As for high-level case, we assume that A_l is provided with a bijection $\widehat{\ }$, with analogous constraints; moreover, we will write $\widehat{A}(v_1, \ldots, v_{ar(A)})$ instead of $\widehat{A(v_1, \ldots, v_{ar(A)})}$.

A *high-level link* over b is a construct $b^d(a)$, where $b \in \mathsf{B}$, $d \in \{+, -\}$ is a *link direction symbol*, and $a \in \mathit{Val} \cup \mathit{Var}$. The set of all high-level links is denoted by B_h and the set of all low-level links is denoted by $\mathsf{B}_l = \{b^d(v) \mid b \in \mathsf{B}, d \in \{+, -\}, v \in \mathit{Val}\}$.

The *deletion* is defined for a multi-set of links $\beta \in \mathcal{M}_f(\mathsf{B}_h)$ and a tie symbol $b \in \mathsf{B}$, as

$$\beta \,\mathbf{del}\, b = \beta \ominus \left(\bigoplus_{l \in \mathcal{M}_f(\{b^d(a) \mid d \in \{+,-\}, a \in \mathit{Var} \cup \mathit{Val}\})} l \right) \quad ,$$

and analogously for low-level links.

A *binding* is a mapping $\sigma\colon \mathit{Var} \to \mathit{Val}$ and an *evaluation* of an entity η (which can be a variable, a vector or a (multi-)set of variables, a set of predicates, a (multi-)set of high-level actions, etc.) through σ is defined as usual and denoted by $\eta[\sigma]$. For instance, if $\sigma = (a_1 \mapsto 2, a_2 \mapsto 3)$, the evaluation of high-level action $A(a_1, a_2, 5)$ through σ is the low-level action $A(2, 3, 5)$. Similarly, the high-level link $b^+(a_1)$ evaluates through σ to the low-level link $b^+(2)$ and the predicate $a_1 = 2$ to *true*.

3 Petri Boxes and M-nets

Petri Boxes are introduced in [4,3,6] as labeled place/transition Petri nets satisfying some constraints, in order to model concurrent systems and programming languages with full compositionality.

Definition 1. *A (low-level) labeled net is a quadruple $L = (S, T, W, \lambda)$, where S is a set of* places*, T is a set of* transitions*, such that $S \cap T = \emptyset$, $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a* weight function*, and λ is a function, called the* labeling *of L, such that:*

- $\forall s \in S\colon \lambda(s) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$ *gives the place status (entry, internal or exit, respectively);*
- $\forall t \in T\colon \lambda(t) = \alpha(t).\beta(t)$ *gives the transition label, where $\alpha(t) \in \mathcal{M}_f(\mathsf{A}_l)$ and $\beta(t) \in \mathcal{M}_f(\mathsf{B}_l)$.*

The behavior of such a net, starting from the *entry marking* (just one token in each e-place), is determined by the usual definitions for place/transition Petri nets.

M-nets are a mixture of colored net features and low-level labeled net ones. They can be viewed as abbreviations of the latter.

Definition 2. *An* M-net *is a triple* (S, T, ι)*, where* S *and* T *are disjoint sets of* places *and* transitions*, and* ι *is an inscription function with domain* $S \cup (S \times T) \cup (T \times S) \cup T$ *such that:*

- *for every place* $s \in S$*,* $\iota(s)$ *is a pair* $\lambda(s).\tau(s)$*, where* $\lambda(s) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$ *is the* label *of* s*, and* $\tau(s) \subseteq \mathit{Val}$*, is the* type *of* s*;*
- *for every transition* $t \in T$*,* $\iota(t) = \lambda(t).\gamma(t)$*, with* $\lambda(t) = \alpha(t).\beta(t)$*, the* label *of* t*, where* $\alpha(t) \in \mathcal{M}_f(\mathsf{AX}_h)$ *is the* action label *and* $\beta(t) \in \mathcal{M}_f(\mathsf{B}_h)$ *is the* link label*;* $\gamma(t)$*, the* guard *of* t*, is a finite set of predicates from* Pr*;*
- *for every arc* $(s, t) \in (S \times T) : \iota((s, t)) \in \mathcal{M}_f(\mathit{Val} \cup \mathit{Var})$ *is a multi-set of variables or values (analogously for arcs* $(t, s) \in (T \times S)$*).*
  $\iota((s, t))$ *will generally be abbreviated as* $\iota(s, t)$*.*

A *marking* of an M-net (S, T, ι) is a mapping $M : S \to \mathcal{M}_f(\mathit{Val})$ which associates to each place $s \in S$ a multi-set of values from $\tau(s)$. In particular, we shall distinguish (like for low-level nets) the *entry marking*, where $M(s) = \tau(s)$ if $\lambda(s) = \mathsf{e}$ and the empty (multi-)set otherwise.

The transition rule specifies the circumstances under which a marking M' is reachable from a marking M. A transition t is *enabled* at a marking M if there is an *enabling binding* σ for variables in the inscription of t (making the guard true) and in arcs around t such that $\forall s \in S : \ \iota(s, t)[\sigma] \leq M(s)$, *i.e.*, there are enough tokens of each type to satisfy the required flow. The effect of an occurrence of t, under an enabling binding σ, is to remove tokens from its input places and to add tokens to its output places, according to the evaluation of arcs' annotations under σ.

The unfolding operation associates a labeled low-level net (see *e.g.* [4]) $\mathcal{U}(N)$ with every M-net N, as well as a marking $\mathcal{U}(M)$ of $\mathcal{U}(N)$ with every marking M of N.

Definition 3. *Let* $N = (S, T, \iota)$*; then* $\mathcal{U}(N) = (\mathcal{U}(S), \mathcal{U}(T), W, \lambda)$ *is defined as follows:*

- $\mathcal{U}(S) = \{s_v \mid s \in S \text{ and } v \in \tau(s)\}$*,*
  and for each $s_v \in \mathcal{U}(S) : \lambda(s_v) = \lambda(s)$*;*
- $\mathcal{U}(T) = \{t_\sigma \mid t \in T \text{ and } \sigma \text{ is an enabling binding of } t\}$*,*
  and for each $t_\sigma \in \mathcal{U}(T) : \lambda(t_\sigma) = \lambda(t)[\sigma]$*;*
- $W(s_v, t_\sigma) = \sum_{a \in \iota(s,t) \,\wedge\, a[\sigma] = v} \iota(s, t)(a)$*, and analogously for* $W(t_\sigma, s_v)$*.*

Let M be a marking of N. $\mathcal{U}(M)$ is defined as follows: for every place $s_v \in \mathcal{U}(S)$, $(\mathcal{U}(M))\,(s_v) = (M(s))\,(v)$. Thus, each elementary place $s_v \in \mathcal{U}(S)$ contains as many tokens as the number of occurrences of v in the marking $M(s)$.

4 Box and M-net algebras

The same operations are defined on both box and M-net levels. They can be divided in two categories: the control flow ones and the communication ones. The first group, which consists of sequential (;) and parallel (‖) compositions, choice (□) and iteration ([∗∗]), can be synthesized from refinement meta-operation [8,9] and they will not be concerned by our extension. The second group concerns the operations which are based on transition composition, and will be directly concerned here, so we introduce them with some details. Only low-level operations are defined formally while we give some intuition for the high-level (M-net) operations. We illustrate the low-level synchronization and restriction on an example and we refer to [6] for further illustrations.

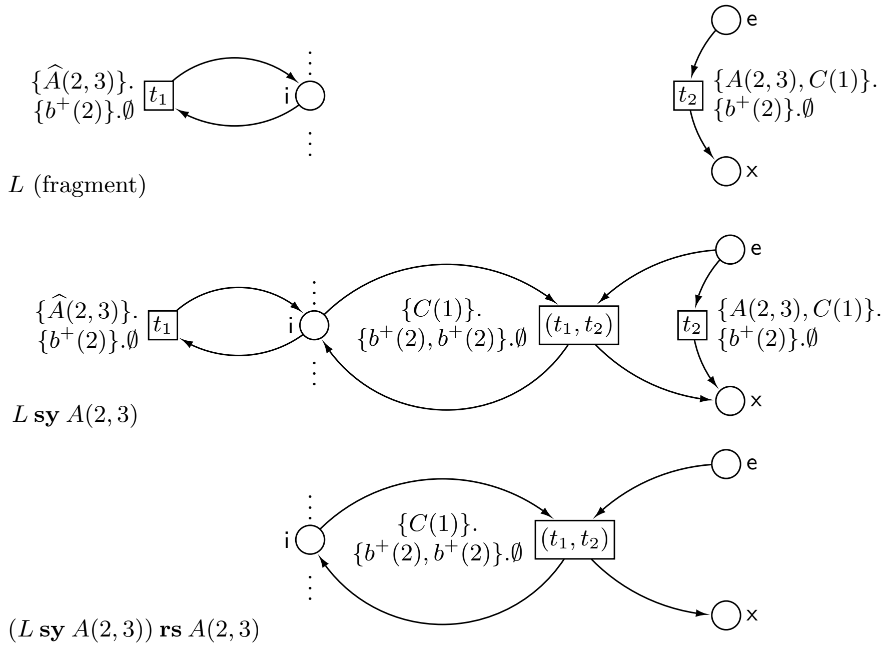
$\{\widehat{A}(2,3)\}.\{b^+(2)\}.\emptyset$
t_1
i
e
t_2
$\{A(2,3), C(1)\}.\{b^+(2)\}.\emptyset$
x
L (fragment)
$\{C(1)\}.\{b^+(2), b^+(2)\}.\emptyset$
(t_1, t_2)
L **sy** $A(2,3)$
$(L$ **sy** $A(2,3))$ **rs** $A(2,3)$

Fig. 1. Synchronization and restriction in PBC.

A synchronization $L\,\mathbf{sy}\,A_l$, with $A_l \in \mathsf{A}_l$, adds transitions to the net L, and can be characterized as CCS-like synchronization, extended to multisets of actions. Intuitively, the synchronization operation of an M-net consists of a repetition of certain basic synchronizations. An example of such a basic synchronization over low-level action $A(2,3)$ of the (fragment of) net L is given in figure 1. Transitions t_1 and t_2 which contain actions $A(2,3)$ and $\widehat{A}(2,3)$ in their labels can be synchronized over $A(2,3)$ yielding a new transition (t_1,t_2). The repetition of such basic synchronizations over $A(2,3)$, for all matching pairs of transitions (containing $A(2,3)$ and $\widehat{A}(2,3)$), yields the synchronization of a net over $A(2,3)$. In M-nets, the actions $A(a_1,a_2)$ and $\widehat{A}(a'_1,a'_2)$ can synchronize through renaming and unification of their parameters.

Definition 4. *Let $L = (S,T,W,\lambda)$ be a low-level net and $A_l \in \mathsf{A}_l$ a low-level action. The synchronization $L\,\mathbf{sy}\,A_l$, is defined as the smallest*[1] *low-level net $L' = (S',T',W',\lambda')$, satisfying:*

- *$S' = S$, $T' \supseteq T$, and $W'|_{(S\times T)\cup(T\times S)} = W$;*
- *if transitions t_1 and t_2 of L' are such that $A_l \in \alpha'(t_1)$ and $\widehat{A_l} \in \alpha'(t_2)$, then L' contains also a transition t with its adjacent arcs satisfying:*
  - $\lambda'(t) = \Big(\alpha'(t_1) \oplus \alpha'(t_2) \ominus \{A_l, \widehat{A_l}\}\Big) . \Big(\beta'(t_1) \oplus \beta'(t_2)\Big)$,
  - $\forall s \in S' \colon W'(s,t) = W'(s,t_1) \oplus W'(s,t_2)$ *and* $W'(t,s) = W'(t_1,s) \oplus W'(t_2,s)$.

The lowest part of figure 1 shows the restriction of $L\,\mathbf{sy}\,A(2,3)$ over the action $A(2,3)$ which returns a net in which all transitions whose labels contain at least one action $A(2,3)$ or $\widehat{A}(2,3)$ are deleted (together with their surrounding arcs). The synchronization followed by the restriction is called *scoping*: $[\![a : L]\!] = (L\,\mathbf{sy}\,a)\,\mathbf{rs}\,a$, for an action a.

5 Asynchronous link operator: tie

In this section, we introduce a new M-net algebra operator, **tie**, devoted to express asynchronous links between transitions. We give first an example in the high-level, and then, define it formally in both high and low-levels.

In figure 2, operator **tie** takes an M-net N and a tie symbol b (we assume that $\mathit{type}(b) = \{1,2\}$), and gives an M-net $N\,\mathbf{tie}\,b$ which is like N but has an additional internal place s_b of the same type as b, and additional arcs between s_b and transitions which carry in their label (high-level)

[1] with respect to the net inclusion, and up to renaming of variables.

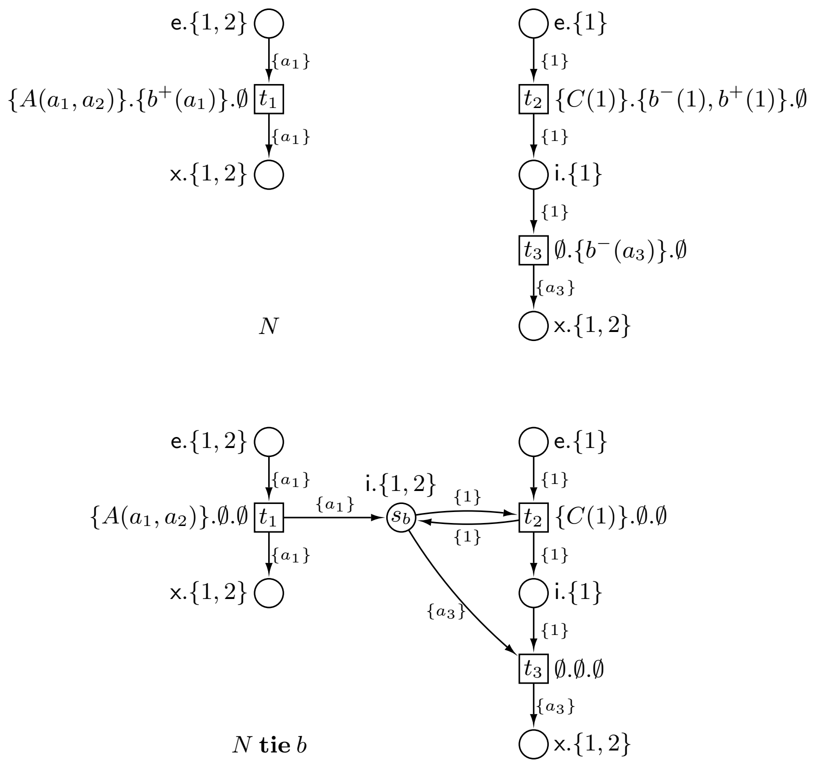

Fig. 2. High-level tie operation.

links over b. The inscriptions of these arcs are (multi-)sets of variables or values corresponding to links over b, and the labels of concerned transitions are as before minus all links over b. For instance, the arc from t_1 to s_b is inscribed by $\{a_1\}$ because there is a link $b^+(a_1)$ in the link label of t_1, which means that the variable a_1 has to be *exported* through b.

Definition 5. *Let* $N = (S, T, \iota)$ *be an M-net and* $b \in \mathsf{B}$ *a tie symbol.* $N\,\mathbf{tie}\,b$ *is an M-net* $N' = (S', T', \iota')$ *such that:*

– $S' = S \uplus \{s_b\}$*, and* $\forall s \in S'\colon \iota'(s) = \begin{cases} \mathsf{i}.\mathit{type}(b) & \text{if } s = s_b, \\ \iota(s) & \text{otherwise}; \end{cases}$

– $T' = T$ *and* $\forall t \in T'\colon \iota'(t) = \alpha(t).(\beta(t)\,\mathbf{del}\,b).\gamma(t)$*,*
if $\iota(t) = \alpha(t).\beta(t).\gamma(t)$*;*

– *$\forall s \in S'$ and $\forall t \in T'$ we have:*
  - $\iota'(s,t) = \begin{cases} \displaystyle\bigoplus_{a \in \mathit{Var} \cup \mathit{Val}} \beta(t)(b^-(a)) \odot a & \text{if } s = s_b, \\ \iota(s,t) & \text{otherwise,} \end{cases}$
  - $\iota'(t,s) = \begin{cases} \displaystyle\bigoplus_{a \in \mathit{Var} \cup \mathit{Val}} \beta(t)(b^+(a)) \odot a & \text{if } s = s_b, \\ \iota(t,s) & \text{otherwise.} \end{cases}$

The tie operation in the low-level is defined similarly. In that case, a place is created for each value in *type*(b) and arcs are added accordingly.

Definition 6. *Let $L = (S, T, W, \lambda)$ be a low-level net and $b \in \mathsf{B}$ a tie symbol. L **tie** b is a low-level net $L' = (S', T', W', \lambda')$ such that:*

– *$S' = S \uplus \{s_{b,v} \mid v \in \mathit{type}(b)\}$, and $\forall s \in S'$: $\lambda'(s) = \begin{cases} \lambda(s) & \text{if } s \in S, \\ \mathsf{i} & \text{otherwise}; \end{cases}$*

– *$T' = T$ and $\forall t \in T'$: $\lambda'(t) = \alpha(t).(\beta(t)\ \mathbf{del}\ b)$, if $\lambda(t) = \alpha(t).\beta(t)$;*

– *$\forall s \in S', \forall t \in T'$ and $\forall v \in \mathit{type}(b)$ we have:*
  - $W'(s,t) = \begin{cases} \displaystyle\sum_{v \in \mathit{Val}} \beta(t)(b^-(v)) & \text{if } s = s_{b,v} \in S' \setminus S, \\ W(s,t) & \text{otherwise,} \end{cases}$
  - $W'(t,s) = \begin{cases} \displaystyle\sum_{v \in \mathit{Val}} \beta(t)(b^+(v)) & \text{if } s = s_{b,v} \in S' \setminus S, \\ W(t,s) & \text{otherwise.} \end{cases}$

6 Properties

Theorem 1. *Let L be a low-level net, $A_l \in \mathsf{A}_l$ and b_1, $b_2 \in \mathsf{B}$. Then:*

1. $(L\ \mathbf{tie}\ b_1)\ \mathbf{tie}\ b_1 = L\ \mathbf{tie}\ b_1$ *(idempotence)*
2. $(L\ \mathbf{tie}\ b_1)\ \mathbf{tie}\ b_2 = (L\ \mathbf{tie}\ b_2)\ \mathbf{tie}\ b_1$ *(commutativity)*
3. $(L\ \mathbf{tie}\ b_1)\ \mathbf{sy}\ A_l = (L\ \mathbf{sy}\ A_l)\ \mathbf{tie}\ b_1$ *(commutativity with synchronization)*

Proof. *1.* By definition 6, operation **tie** makes desired links and removes concerned tie symbols from the transitions labels. A second application of **tie** over the same tie symbol does not change anything in the net.

2. By definition 6, operations **tie** for different tie symbols are totally independent, the order of applications has no importance.

3. By definition 6 and 4, when **tie** is applied first, it creates arcs which are inherited by the new transitions created by **sy**. Conversely, if **sy** is applied first, it transmits links to the new transitions, allowing **tie** to create the expected arcs.

Since operation **tie** is commutative, it naturally extends to a set of tie symbols.

Theorem 2. *Let N be an M-net, and $b \in \mathsf{B}$. Then:*

$$\mathcal{U}(N\ \mathbf{tie}\ b) = \mathcal{U}(N)\ \mathbf{tie}\ b.$$

Proof. It is enough to remark that the high-level tie operation creates a place s_b with type $type(b)$ and adds arcs to/from transitions which carry links on b in their inscriptions. The unfolding gives for this new place a set of places $\{s_{b,v} \mid v \in type(b)\}$, and the set of arcs to/from transitions t_σ. The weight of an arc between place $s_{b,v}$ and transition t_σ corresponds to the multiplicity of value v in the evaluation through σ of the high-level arc inscription between s_b and t. By definition, this is exactly what is done by the low-level tie operation.

Now, corollary 1 comes directly from the two above theorems and from the commutativity of synchronization with unfolding [6] (notice that, after the introduction of links, it is obvious that this commutativity is preserved).

Corollary 1. *Let N be an M-net, $A_h \in \mathsf{A}_h$ and b_1, $b_2 \in \mathsf{B}$. Then:*

1. $(N\ \mathbf{tie}\ b_1)\ \mathbf{tie}\ b_1 = N\ \mathbf{tie}\ b_1$ *(idempotence)*
2. $(N\ \mathbf{tie}\ b_1)\ \mathbf{tie}\ b_2 = (N\ \mathbf{tie}\ b_2)\ \mathbf{tie}\ b_1$ *(commutativity)*
3. $(N\ \mathbf{tie}\ b_1)\ \mathbf{sy}\ A_h \equiv (N\ \mathbf{sy}\ A_h)\ \mathbf{tie}\ b_1$ *(commutativity with synchronization)*

   where $\equiv$ identifies M-nets which are equivalent modulo renaming of variables [6].

7 An application to discrete time modeling

The newly introduced **tie** operator has an immediate application in a modeling of discrete time within M-nets. A clock is built over principles described in [10,15]: an arbitrary event (*i.e.*, a transition occurrence) is counted and used to build the time scale which all the system refers to. Two points can be puzzling: the time scale is not even and the clock can be frozen to allow the system meeting its deadlines. Both these points are discussed in [10] and are shown to be irrelevant in the context of Petri nets used for *specification* (by opposition to *programming*).

Our clock (which actually is a server) is implemented by the M-net N_{clock} depicted in figure 3. It can handle concurrent *counting requests* for different parts of the system: each request is assigned an identifier which is allocated by the clock on a *start* action; next, this identifier can be used to perform *check* actions which allows to retrieve current pulse-count for the request; finally, a *stop* can delete the request, sending back the pulse-count again.

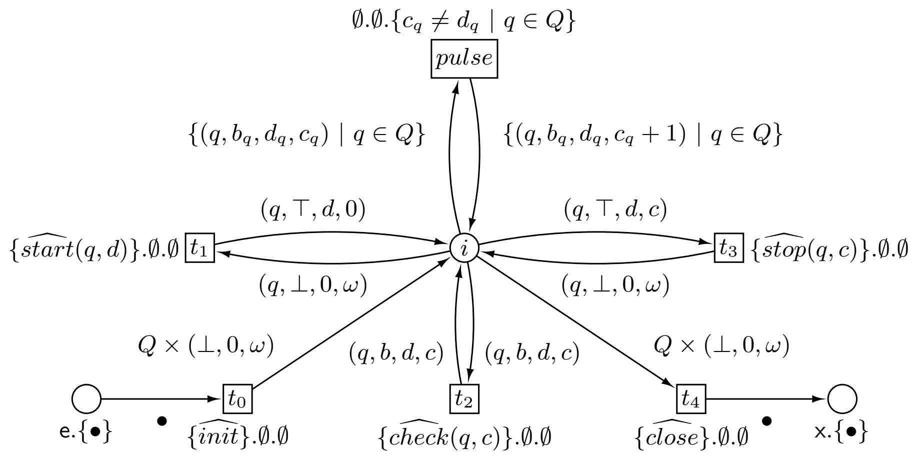
$\emptyset.\emptyset.\{c_q \neq d_q \mid q \in Q\}$
$pulse$
$\{(q, b_q, d_q, c_q) \mid q \in Q\}$
$\{(q, b_q, d_q, c_q + 1) \mid q \in Q\}$
$(q, \top, d, 0)$
$(q, \top, d, c)$
$\{\widehat{start}(q, d)\}.\emptyset.\emptyset$
t_1
i
t_3
$\{\widehat{stop}(q, c)\}.\emptyset.\emptyset$
$(q, \bot, 0, \omega)$
$(q, \bot, 0, \omega)$
$Q \times (\bot, 0, \omega)$
(q, b, d, c)
(q, b, d, c)
$Q \times (\bot, 0, \omega)$
t_0
t_2
t_4
e.{•}
•
$\{\widehat{init}\}.\emptyset.\emptyset$
$\{\widehat{check}(q, c)\}.\emptyset.\emptyset$
$\{\widehat{close}\}.\emptyset.\emptyset$
•
x.{•}

Fig. 3. The M-net N_{clock}: the place i is labeled $\mathsf{i}.Q \times \{\bot, \top\} \times \widetilde{\mathbb{N}} \times \widetilde{\mathbb{N}}$.

The clock N_{clock}, depicted in figure 3, works as follows:

- it starts when the transition t_0 fires and put tokens in the central place of N_{clock}, each token being a tuple (q, b, d, c), corresponding to a request, where $q \in Q$ is the identifier for the request, $b \in \{\bot, \top\}$ tells if the request is idle ($b = \bot$) or in use ($b = \top$), d is the maximum number of pulse to count for the request and c is the current pulse-count. Both d and c are values in $\widetilde{\mathbb{N}} = \mathbb{N} \cup \omega$ where ω is greater than any integer and $\omega + 1 = \omega$;
- a request begins when t_1 fires. *start* has two parameters: the request identifier, q, is sent to the client which provides the duration, d, as the maximum pulse-count for the request;
- next, the current pulse-count can be *check*ed with transition t_2: the identifier is provided by the client which gets the current pulse-count returned back through c;

– t_3 is used to *stop* a request, it acts like t_2 but also sets the request's token idle;
– the clock can be terminated with transition t_4;
– at any time, provided its guard is true, *pulse* can fire, incrementing the pulse-count for *all* requests. This explains the idle value $(q, \bot, 0, \omega)$ for the tokens in i: it does neither change after a pulse since $\omega = \omega + 1$ nor affect the guard on *pulse* because $\omega \neq 0$. This guard is used to prevent *pulse* from firing if a request is about to end (pulse-count is equal to the maximum announced on *start*), this ensures a timed sub-net will always meet its deadline if it has been announced on *start*.

In order to use this clock, one just has to add “clock actions” (*start*, *check* and *stop*) on the transitions intended to be timed and asynchronous links should be used to transport the request identifiers between a *start* and the corresponding *check*(s) and *stop*(s), like in figure 4.

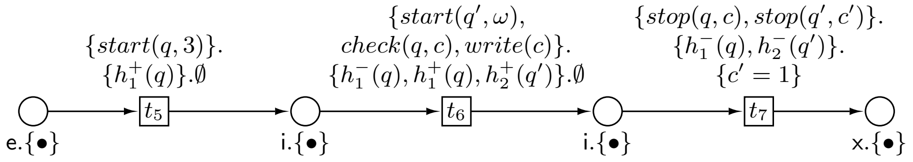
$\{start(q,3)\}.$
$\{h_1^+(q)\}.\emptyset$
$\{start(q',\omega),$
$check(q,c), write(c)\}.$
$\{h_1^-(q), h_1^+(q), h_2^+(q')\}.\emptyset$
$\{stop(q,c), stop(q',c')\}.$
$\{h_1^-(q), h_2^-(q')\}.$
$\{c'=1\}$
t_5
t_6
t_7
e.{●}
i.{●}
i.{●}
x.{●}

Fig. 4. An example of time-constrained M-net.

In the net of figure 4, we specify the following constraints:

– the $start(q, 3)$ on t_5 corresponds to the $stop(q, c)$ on t_7 (thanks to the links on h_1) so there can be at most 3 pulses between t_5 and t_7 because of the guard on *pulse*;
– similarly, there must be exactly 1 pulse between t_6 and t_7 (here we use the links on h_2 to transmit the identifier) but the constraint is expressed through the guard on t_7. In this case, since the deadline was not announced in the start, it is possible to have a deadlock if the system runs too slow, *i.e.*, does not meet its deadline;
– on t_6, we use a *check* for the request started on t_5 (the identifier is imported *and* exported on h_1) to retrieve the pulse count between t_5 and t_6. This count is sent to another part of the system with the action $write(c)$ which should be synchronized with a $\widehat{write}$, in a piece of the specification not represented here.

8 Conclusion

We presented an extension of M-nets and PBC in order to cope with asynchronous communications at the (net) algebra level. This extension led to a simple and elegant mean to model discrete time within M-nets, so we hope it would be useful to work with timed specifications. Moreover, we could expect to apply asynchronous links for a wider range of applications. In particular, we yet study: real-time programming with B(PN)^2 [7] (a parallel programming language with M-nets and PBC semantics), discrete timed automata simulation and a new implementation of B(PN)^2's FIFO channels.

References

1. V. Benzaken, N. Hugon, H. Klaudel, E. Pelz, and R. Riemann. M-net Based Semantics for Triggers. *ICPN'98* LNCS Vol. 1420 (1998).
2. E. Best. A Memory Module Specification using Composable High Level Petri Nets. Formal Systems Specification, The RPC-Memory Specification Case Study, LNCS Vol. 1169 (1996).
3. E. Best, R. Devillers, and J. Esparza. General Refinement and Recursion for the Box Calculus. *STACS'93*. Springer, LNCS Vol. 665, 130–140 (1993).
4. E. Best, R. Devillers, and J.G. Hall. The Box Calculus: a New Causal Algebra with Multilabel Communication. *APN'92*. Springer, LNCS Vol. 609, 21–69 (1992).
5. E. Best and H. Fleischhack, editors. *PEP: Programming Environment Based on Petri Nets*. Number 14/95 in Hildesheimer Informatik-Berichte. Univ. Hildesheim (1995).
6. E. Best, W. Fraczak, R.P. Hopkins, H. Klaudel, and E. Pelz. M-nets: an Algebra of High Level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages. *Acta Informatica:35* (1998).
7. E. Best, R.P. Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notation. *PARLE'93*. Springer, LNCS Vol. 694, 379–390 (1993).
8. R. Devillers and H. Klaudel. Refinement and Recursion in a High Level Petri Box Calculus. *STRICT'95*. Springer, ViC, 144–159 (1995).
9. R. Devillers, H. Klaudel and R.-C. Riemann. General Refinement for High Level Petri Nets. *FST & TCS'97*, Springer, LNCS Vol. 1346, 297–311 (1997).
10. R. Durchholz. Causality, time, and deadlines. *Data & Knowledge Engineering*, 6:496–477 (1991).
11. H. Fleischhack and B. Grahlmann. A Petri Net Semantics for B(PN)^2 with Procedures. *Parallel and Distributed Software Engineering*, Boston Ma. (1997).
12. H. Klaudel and R.-C. Riemann. Refinement Based Semantics of Parallel Procedures. In proceedings of *PDPTA'99* (1999).
13. M. Koutny, J. Esparza, E. Best. Operational Semantics for the Petri Box Calculus. *CONCUR'94*. Springer, LNCS Vol. 836, 210–225 (1994).
14. J. Lilius and E. Pelz. An M-net Semantics for B(PN)^2 with Procedures. *ISCIS XI*, Antalya, Vol. I, 365–374 (1996).
15. G. Richter. Counting interfaces for discrete time modeling. Technical Report repset-1998-26, GMD – German National Research Center for Information Technology, SET (1998).

A Specifying a railroad crossing with timed M-nets

We intend to specify a system with the following characteristics:

- it has n parallel and one-way tracks;
- each track has two sensors, one to detect when a train approaches (it raises the signal *app*) and the other to detect when the train leaves (then it raises *far*);
- when a train activates the first sensor, it takes between 3 and 5 time units to reach the gates, then it takes 4 to 6 time units to leave the second sensor (which is activated at this moment);
- gates go up when they receive the signal *up* and go down on the occurrence of *down*. The latter signal is taken in consideration as soon as emitted (this may cause a direction change) while the former is delayed to the next time the gates will be down;
- the gates take 3 time units to raise or lower fully;
- trains are distant enough: for one track, there cannot be two *app* without a *far* between;
- neither trains nor gates are supposed to break down.

Signals are implemented with actions. The system's specification is an M-net N_{system} built from the sub-nets $N_{control}$ for the controller, N_{gates} for the gates and N^i_{track} $(1 \leq i \leq n)$ for ith railway. We define

$$N_{system} = \left[\!\!\left[\{app, far, down, up\} \;:\; N_{control} \| N_{gates} \| N^1_{track} \| \cdots \| N^n_{track} \right]\!\!\right] \;,$$

so the complete specification can be obtained as:

$$\left[\!\!\left[\{init, start, check, stop, close\} : N_{clock} \;\Big\|\; (N_{init}; N_{system}; N_{close}) \right]\!\!\right] \quad ,$$

where N_{clock} is the clock from the figure 3 and N_{init} and N_{close} are single-transition M-nets with the inscriptions $\{init\}.\emptyset.\emptyset$ and $\{close\}.\emptyset.\emptyset$, respectively.

In the following, all the figures are simplified: if not specified, places have type $\{\bullet\}$ and arcs carry $\{\bullet\}$; places with name e are entry places and places x are exit places, any other place is internal. We also omit $\emptyset.\emptyset.\emptyset$ annotations for some transitions.

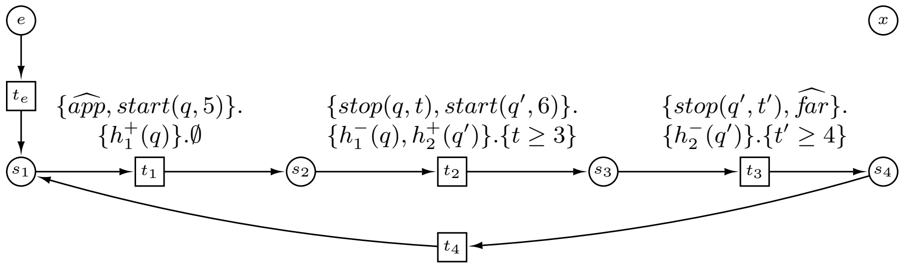
e
x
t_e
$\{\widehat{app}, start(q,5)\}.$
$\{h_1^+(q)\}.\emptyset$
$\{stop(q,t), start(q',6)\}.$
$\{h_1^-(q), h_2^+(q')\}.\{t \geq 3\}$
$\{stop(q',t'), \widehat{far}\}.$
$\{h_2^-(q')\}.\{t' \geq 4\}$
s_1
t_1
s_2
t_2
s_3
t_3
s_4
t_4

Fig. 5. The M-net N'^{i}_{track} used to specify the ith track.

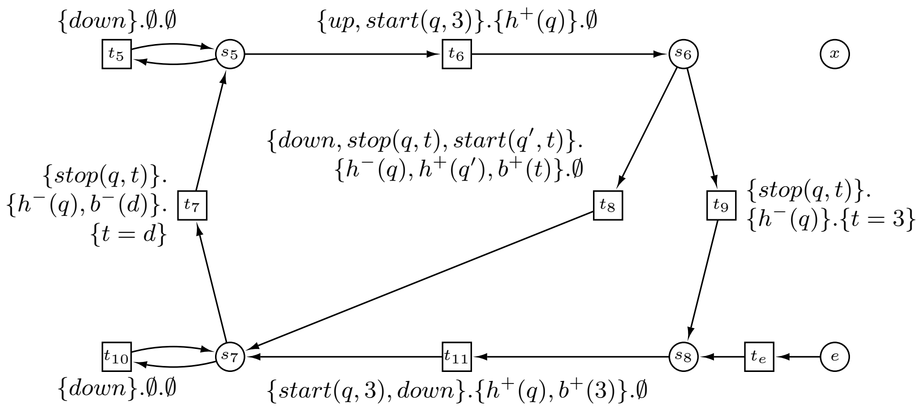
$\{down\}.\emptyset.\emptyset$
$\{up, start(q,3)\}.\{h^+(q)\}.\emptyset$
t_5
s_5
t_6
s_6
x
$\{down, stop(q,t), start(q',t)\}.$
$\{h^-(q), h^+(q'), b^+(t)\}.\emptyset$
$\{stop(q,t)\}.$
$\{h^-(q), b^-(d)\}.$
$\{t = d\}$
t_7
t_8
t_9
$\{stop(q,t)\}.$
$\{h^-(q)\}.\{t = 3\}$
t_{10}
s_7
t_{11}
s_8
t_e
e
$\{down\}.\emptyset.\emptyset$
$\{start(q,3), down\}.\{h^+(q), b^+(3)\}.\emptyset$

Fig. 6. The M-net N'_{gates} used to specify the gates.

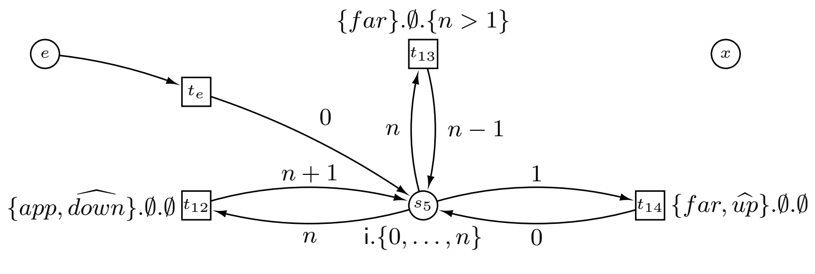
$\{far\}.\emptyset.\{n > 1\}$
e
t_{13}
x
t_e
0
n
$n-1$
$n+1$
1
$\{app, \widehat{down}\}.\emptyset.\emptyset$
t_{12}
s_5
t_{14}
$\{far, \widehat{up}\}.\emptyset.\emptyset$
n
$\mathsf{i}.\{0,\ldots,n\}$
0

Fig. 7. The M-net $N_{control}$ which specifies a simplified version of the controller.

A.1 Tracks

All tracks N^i_{track} $(1 \leq i \leq n)$ are identical: $N^i_{track} = N'^i_{track}$ **tie** $\{h_1, h_2\}$ where N'^i_{track} is the M-net depicted in figure 5. We also define $type(h_1) = type(h_2) = Q$ where Q is the set of all request identifiers.

Places and transitions can be interpreted as follows:

- s_1, a train is far away, before the crossing;
- t_1, the train reaches the first sensor which raises the *app* signal. This also starts a counting request for 5 pulses at most (*i.e.*, the maximum needed for the train to reach the gates);
- s_2, the train is between the first sensor and the gates;
- t_3, the train reaches the gates, the guard ensures it took at least 3 pulses. Another counting request is started to specify the time until the train leaves;
- s_3, the train is in the gates or between the gates and the second sensor;
- t_3, the train leaves the second sensor which raises the signal *far*;
- s_4, the trains has left the area of study;
- t_4 is used to "produce" a new train on the same track.

We can notice that this M-net, like other parts of the system, runs endless because the system itself should be endless.

A.2 The gates

Like for the tracks, we use an auxiliary M-net shown in figure 6 and assume that $type(h) = Q$ and $type(b) = \{0, \ldots, 3\}$. With this net, we define: $N_{gates} = N'_{gates}$ **tie** $\{h, b\}$.

The net starts with the transition t_i, holding a token in s_8 which means that the gates are open. Then, a standard run is a loop on t_{11}, s_7 (the gates are going down), t_7, s_5 (the gates are down), t_6, s_6 (the gates are going up), t_9 and return to s_8. Since a *down* signal can occur while the gates are opening, we need transition t_8 which implies a backward movement. When such an interruption occurs, we specify the fact that the gates are not completely open by retrieving the time passed since the last *up* signal. This time is t from action $stop(q, t)$ on t_8 and it is exported over the link symbol b. So, t_7 can import it to have a valid guard: if the gates raise during d time units, they need exactly d to go down. In the standard looping run, t_{11} exports 3 which is the time required for a full coming down.

This example shows that asynchronous links are useful not only for request identifiers but also for other purpose: here, we transmit the duration for the raising/lowering of the gates through an asynchronous link and it leads to a smaller M-net which focuses on the control flow while the less interesting information is hidden in links.

A.3 The controller

The specifications of the controller does not need asynchronous links so we restrict it to the minimum because it is less important for the example. Our simplification leads to a sequencing of signal receptions, but since the parallel version of the controller is actually more complex, we prefer to restrict ourselves to what is given here.

The (simplified) controller is depicted in figure 7. After the initialization, the place s_9 holds the token 0 which means that there is no train in the area of study (*i.e.*, between the two sensors on each track). Each time an *app* signal is received, it is translated into a *down* signal for the gates and the number of trains in s_9 is increased by 1 (transition t_{12}). When a *far* is received, two situations are possible: if there is 1 in s_9, we know that the last train has just left and the signal *up* is emitted while the train-count is set to 0 (transition t_{14}); if there is more than one train, transition t_{13} just decreases the train-count and no *up* is relayed.