M-nets: A survey
Hanna Klaudel, Franck Pommereau

To cite this version:

HAL Id: hal-00870484
https://hal.science/hal-00870484
Submitted on 8 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Hanna Klaudel · Franck Pommereau

M-nets: a survey.

the date of receipt and acceptance should be inserted later

Abstract This paper surveys the research related to the model of M-nets since it was introduced in 1995. M-nets are high-level labelled Petri nets which can be composed, like process algebra terms, using various operators. We present the core model, several of its extensions and the main applications.

Keywords High-level Petri nets, compositions, action refinement, synchronisation.

1 Introduction

The model of Petri nets [58] is based on concepts coming from automata theory, linear algebra and graph theory. Besides the general advantages of a formal model and the verification methods based on linear algebra, Petri nets are additionally attractive since they have a simple graphical representation. This characteristic is important already in the design process showing how a concurrent system is built and distributed in space. It gives a clear image of concurrency, sequentiality and conflict, both on the concrete visual level and on the graph-theoretic level. In particular, such an explicit representation of concurrency is suitable when studying non sequential ("true concurrency" or partial order) semantics of concurrent systems.

On the other hand, the modular design of large systems allows to reduce and manage their complexity. This is possible either in a bottom-up way by composing smaller subsystems, or going top-down by refining parts of a rough model by more detailed system descriptions. In both cases, systems
are constructed from building blocks and a semantics should support the modular construction of systems. Modularity has been a somewhat weak point of Petri net theory: a Petri net is defined as a whole, and not in the first instance obtained by composing subnets. This is totally different in process algebras where systems are described by process terms, which are by definition built from subterms. The semantics of a process term is obtained from the semantics of its subterms and is compositional by nature. However, the standard process algebras reduce concurrency to interleaving arguing that interleaving is simpler than true concurrency and just as expressive. On the other hand, interleaving based models are less suitable for a top-down design, since they describe systems using actions that are assumed to be instantaneous and indivisible.

Both Petri nets and process algebras approaches have increasingly influenced each other, and considerable effort has been made to combine their respective merits. The Petri Box Calculus (PBC) [4, 5], which evolved later to the Petri Net Algebra (PNA) [6], offers a very general solution to combine process algebras and Petri nets. However, in practical situations, the standard low-level Petri net model on which PBC is based leads to huge nets which are well defined mathematically but difficult to grasp intuitively. As a consequence, the appreciated feature of Petri nets of representing graphically system properties such as concurrency, sequentiality, conflicts, etc., vanishes because of the size of Petri nets necessary to describe the considered system. To address this problem, high-level Petri net models have been proposed, such as predicate/transition nets [42], coloured Petri nets [47], or algebraic nets [76].

In particular, the Calculus of Modular Multilabelled Nets (M-nets) [8, 51] was introduced in order to combine the compositionality of process algebras and the explicit representation of concurrency of Petri nets in a common high-level framework. Indeed, M-nets are considered as the coloured counterpart of the PBC. Actually, M-nets and PBC are related through an operation of unfolding which takes an M-net $N$ and yields an equivalent low-level net $N_I$. $N$ can be seen as an abbreviation of $N_I$, and $N_I$ as the semantics of $N$. M-nets support various composition operations (parallel, sequence, choice, iteration, synchronisation, restriction, etc.), which are essentially the same as in PBC. Indeed, it was one of the main aims in the design of the M-net model to ensure that the unfolding of a composed net coincides with the composition of the unfoldings of its parts. The PBC and M-nets are implemented in the PEP tool [41] which allows to edit, simulate and verify systems using model-checking. Moreover, the SNAKES toolkit [73], a tool specifically dedicated to work with variants of M-nets and PBC, allows to quickly implement new operations for these models, thus providing a framework to prototype and experiment with new variants.

This paper surveys the research related to M-nets since they were introduced. The next section defines the Petri net aspects of the model, including the refinement (meta-)operation which allows to substitute M-net transitions by arbitrary M-nets. The low-level net model is also introduced in order to state the consistency between high and low levels. Section 3 presents several extensions of the core model: the parameterised refinement allowing to
exchange information between different abstraction levels of a system; recursion which is like a repetitive refinement; and buffered communication providing a simple scheme to share data between different parts of an M-net. Section 4 shows how an algebra of M-nets is built using their annotations (for synchronous and asynchronous communication) and the refinement (for the control flow operations). The introduced operations are consistent with those existing in PBC. Section 5 reviews several applications or further extensions of the M-nets algebra. This includes a definition of an M-net semantics of the parallel specification language B(PN)$^2$ and several other extensions like the introduction of object oriented paradigms, the modelling of mobility, pre-emption (suspend/resume and abort of subsystems) or timing constraints. Finally, section 6 presents a case study using M-nets to model and verify a timed railroad crossing system.

2 The model of M-nets

2.1 Basic definitions and notations

We start with the definition of multisets which are widely used in the following. Let $E$ be a set. A multiset over $E$ is a function $\mu : E \rightarrow \mathbb{N}$ which associates to each element of $E$ its number of occurrences in $\mu$. A multiset $\mu$ is finite if so is the set $\{x \in E \mid \mu(x) \neq 0\}$. Sometimes, we will use for multisets an extended set notation; for instance, $\{x, y, y\}$ will denote the multiset $\mu$ such that $\mu(x) = 1$, $\mu(y) = 2$ and $\mu(z) = 0$ for all $z \in E \setminus \{x, y\}$. We will denote by $\emptyset$ the empty multiset. We also introduce the following notations for $\mu$, $\mu_1$ and $\mu_2$ multisets over $E$ and $n \in \mathbb{N}$:

- the symbols $+, -, \ast$, and $\ast$ denote, respectively, the sum, the difference of multisets and the multiplication of a multiset by a natural number; formally, for $x$ in $E$: $(\mu_1 + \mu_2)(x) \equiv \mu_1(x) + \mu_2(x)$, $(\mu_1 - \mu_2)(x) \equiv \max(0, \mu_1(x) - \mu_2(x))$ and $(n \ast \mu)(x) \equiv n \ast \mu(x)$;
- we write $x \in \mu$ if $\mu(x) > 0$, and $\mu_1 \leq \mu_2$ if $\forall x \in E : \mu_1(x) \leq \mu_2(x)$;
- we denote by $\text{mult}(E)$ the set of all the multisets over $E$, and by $\text{mult}_f(E)$ the set of the finite multisets over $E$.

A (low-level) Petri net is a directed bipartite graph whose nodes are places or transitions. The places may be marked, i.e., may carry tokens. The input arcs of a transition $t$ come from the set of pre-places of $t$ and the output arcs of $t$ go to its post-places. A transition is activated if its pre-places are marked by sufficiently many tokens. It may then be fired in which case some tokens are removed from each pre-place and some other produced in each post-place. The arcs are weighted, i.e., annotated by a natural number (where 0 corresponds to the absence of the arc) indicating the number of tokens which are transported through the arc during the firing of a transition. Places and transitions may also carry labels. Formally:

**Definition 1** A (low-level) labelled Petri net $N$ is a quadruple $(S, T, W, \lambda)$ where:

- $S$ is a set of places and $T$ is a set of transitions, with $S \cap T = \emptyset$;
- $W : (S \times T) \cup (T \times S) \rightarrow \mathbb{N}$ is the weight function on arcs;
- $\lambda$ is the labelling function on $S \cup T$.

A marking of $N$ is a function $M : S \rightarrow \mathbb{N}$ which associates to each place the number of tokens it carries.

For a place or a transition $x \in S \cup T$, we define $\bullet x \overset{\text{def}}{=} \{y \in S \cup T \mid W(y, x) > 0\}$ and $x^* \overset{\text{def}}{=} \{y \in S \cup T \mid W(x, y) > 0\}$. The marking of a place defines a local state, in such a way that the global state of the net is represented by the set of all such local states. The dynamic behaviour of such a net is given by the transition rule:

**Definition 2** Let $N = (S, T, W, \lambda)$ be a labelled Petri net and $M$ its marking. A transition $t \in T$ is **activated** at $M$ iff $\forall s \in S : M(s) \geq W(s, t)$. The **firing** of $t$ produces the visible action $\lambda(t)$ and gives rise to the new marking $M'$ defined by $\forall s \in S : M'(s) \overset{\text{def}}{=} M(s) - W(s, t) + W(t, s)$.

The transition rule illustrates the property of locality of nets: only the part $\bullet t^* \cup t^*$ of the global state is involved in the firing of the transition $t$. In concurrent systems, the actions (represented by the occurrences of transitions) may appear concurrently, i.e., independently of each other. If these occurrences are described by an arbitrary interleaving of actions, then each sequence of independent actions is a sequence of occurrences of the system and the corresponding semantics is called a **sequential** or an **interleaving** semantics. If the occurrence of a finite multiset of actions is allowed, then the corresponding semantics is a **step** or a concurrent semantics [3,9,43,44]. If the occurrences of actions are partially ordered, then the corresponding semantics is a **partial order** or **true concurrency** semantics [66,79,81].

The model based on low-level nets is particularly interesting because it is supported by various implemented tools (see [69,80] for a presentation of many tools) and may be analyzed using methods devoted to such nets or using efficient algorithms of model checking. However, these nets are not often used directly because the specifications of real size systems are in general too large to be understandable. The designers often prefer to use high-level versions of Petri nets which provide a better abstraction and which may be automatically **unfolding** to low-level nets before being analysed.

Figure 1 represents two equivalent nets (in the sense that they describe the same behaviours), the net on the left is high-level and the net on the right is its unfolding (so, it is a low-level net).

The annotations used for the high-level nets have the following meanings:

- the places are **typed**, i.e., have associated sets of values (s has the type $\{1, 2, 3\}$ and $s'$ has the type $\{2, 3\}$);
- the tokens are values respecting the types of the places ($s'$ carries the token 2 and $s''$ carries the token 3);
- the arcs are annotated with **values** or **variables**: the scope for the variables is bound to a transition and its adjacent arcs (so, the variables around a transition have only a local meaning and may be consistently renamed);
- the transitions may carry Boolean expressions called **guards** playing the role of firing conditions: a transition can fire only with tokens which make
the guard true (this is the case for the transition $t$ taking, for instance, $x = 2$ and $y = 3$, but not for $x = y = 2$). The guard is often used in order to compute values during a firing. Here, since $x$ and $y$ are free variables (not bound on an input arc), the transition actually computes non-deterministically a number in $\{1, 2, 3\}$ for $x$ and another one in $\{2, 3\}$ for $y$, choosing them distinct (because of the guard).

The transition rule in a high-level net is more complicated than that used at the low-level: in order to fire a transition $t$ it is necessary to take tokens in the pre-places of $t$, to associate them to the variables around $t$ and to ensure that the guard evaluates to true. Moreover, only values allowed by the types of places may be produced. The mappings, called bindings, associating a value to each variable exist independently of the marking of the net and of the guard of $t$. (However, we shall only consider the variables involved in $t$ when showing a binding intended to be used for $t$.) For $t$ in figure 1, the bindings are of the form $(x \rightarrow i, y \rightarrow j)$ where $i$ and $j$ are arbitrary values. A binding is called a mode if it allows to evaluate the guard to true and if the value corresponding to each variable appearing in an annotation of an arc belongs to the type of the adjacent place. So, the modes of $t$ in figure 1 are $\sigma_1 \triangleq (x \rightarrow 1, y \rightarrow 2)$, $\sigma_2 \triangleq (x \rightarrow 1, y \rightarrow 3)$, $\sigma_3 \triangleq (x \rightarrow 2, y \rightarrow 1)$ and $\sigma_4 \triangleq (x \rightarrow 3, y \rightarrow 2)$, but not $\sigma'_1 \triangleq (x \rightarrow 1, y \rightarrow 1)$ nor $\sigma'_2 \triangleq (x \rightarrow 2, y \rightarrow 2)$ because they do not respect the guard of $t$, nor $\sigma'_3 \triangleq (x \rightarrow 0, y \rightarrow 2)$ because it does not respect the type of $s'$.

These high-level aspects are expressible in the low-level nets through the operation of unfolding defined as follows:

- each high-level place is unfolded to as many low-level places as there are values in its type (so, $s'$ gives rise to the low-level places $s'_1$, $s'_2$ and $s'_3$);
- the marking is obtained in such a way that, for instance, the token 2 in $s'$ leads to a (black) token in the place $s'_2$;
- each high-level transition is unfolded to as many low-level transitions as there are modes associated to it (so, $t$ generates the low-level transitions $t_{s'_1}$ to $t_{s'_4}$);
- the arcs are obtained consistently with the modes. So, according to the mode $\sigma_1 = (x \rightarrow 1, y \rightarrow 2)$, $t_{s'_1}$ is connected to $s'_1$, with the weight 1.

**Fig. 1** A high-level net (on the left) and its unfolding (on the right) whose labelling has been omitted. The $\sigma_i$'s are the modes of $t$ in the high-level net, the indexes of the $s'_i$'s and the $s''_i$'s correspond to the values in the types of $s'$ and $s''$. 

\[
\begin{align*}
\text{\{1, 2, 3\}} & \rightarrow \{2, 3\} \\
\bullet & \rightarrow \bullet \\
\end{align*}
\]
corresponding to the multiplicity of \( x \) in the annotation of the high-level arc, and to \( s^2_j \), with the weight 1 corresponding to the multiplicity of \( y \) in the annotation of the high-level arc.

The unfolding allows to express systems in the high-level domain with the guarantee that they have a representation in the low-level one, which is needed for their automated analysis. (Notice, however, that allowing infinite types for the high-level places gives rise to infinite low-level nets.)

2.2 Static and dynamic aspects of M-nets

We consider the following pairwise disjoint sets:

- \( \text{Val} \) is the set of values (in particular, \( \text{Val} \) contains the “black token” \( \bullet \), natural numbers, Boolean values, etc.);
- \( \text{Var} \) is the set of variables (\( \text{Var} \) is assumed large enough to allow renaming each time it is necessary in order to avoid name clashes);
- \( \mathcal{A} \) is the set of high-level action symbols, provided with a bijection \( \hat{\ } \), called conjugation, such that for all \( \text{act} \in \mathcal{A} : \hat{\text{act}} \neq \text{act} \) and \( \hat{\hat{\text{act}}} = \text{act} \).
- \( \mathcal{A}_e \) corresponds to \( \mathcal{A} \) in the low-level domain; it is the set of all elementary actions constructed from \( \mathcal{A} \) and \( \text{Val} \). (Notice that \( \mathcal{A}_e \) is closed under \( \hat{\ } \).
- \( \mathcal{X} \) contains the hierarchical symbols, which will be used to denote abstract actions used to label transitions to be refined;
- the symbols \( e, i \) and \( x \) denote the status of places used to label places in order to guide net compositions; for a net \( N \), we denote by \( N^e \), \( N^i \) and \( N^x \), respectively, the set of its entry places (labelled by \( e \)), internal places (labelled by \( i \)) and exit places (labelled by \( x \)).

The boxes are labelled low-level nets with some structural constraints: \(^1\) the entry places have no input arcs (symmetrically, the exit places have no output arcs), there always exists entry and exit places and each transition has at least one pre-place and one post-place.

**Definition 3** A box \( N = (S, T, W, \lambda) \) is a low-level labelled Petri net such that:

- for each place \( s \in S \) we have \( \lambda(s) \in \{ e, i, x \} \);
- for each transition \( t \in T \), we have \( \lambda(t) \in \text{mult} (\mathcal{A}_e) \) (if \( t \) is a communication transition) or \( \lambda(t) \in \mathcal{X} \) (if \( t \) is a hierarchical transition);
- \( N \) is ex-restricted: \( N^e \neq \emptyset \neq N^x \);
- \( N \) is ex-oriented: \( \forall t \in T, \forall s \in N^e, \forall s' \in N^x : W(t, s) = 0 \land W(s', t) = 0 \);
- \( N \) is \( T \)-restricted: \( \forall t \in T, \exists s, s' \in S : W(s, t) > 0 \land W(t, s') > 0 \).

\(^1\) These conditions were required in PBC [4] but relaxed in PNA [6]. We will use the original definition which leads to a simpler, more intuitive, model.
With this definition, in particular, we are sure to have non-empty entry and exit interfaces ($N^e$ and $N^x$) in every box $N$. The communication and the hierarchical interfaces, which are composed of transitions, may be empty. These interfaces will be crucial for defining composition operations on boxes, see section 4. The transition rule for boxes is that of low-level labelled Petri nets.

$M$-nets are high-level boxes. In order to introduce them, we need essentially to enrich the annotations of low-level nets.

**Definition 4** An $M$-net is a triple $N = (S, T, \iota)$, where $S$ is the set of places, $T$ is the set of transitions (with $S \cap T = \emptyset$) and $\iota$ is the annotation function on $S \cup T \cup (S \times T) \cup (T \times S)$, such that:

- for each place $s \in S$, $\iota(s)$ is a pair $\lambda(s), \alpha(s)$ where $\lambda(s) \in \{e, i, x\}$ gives the status of $s$ and $\alpha(s) \subseteq \text{Val}$, with $\alpha(s) \neq \emptyset$, gives its type;
- for each transition $t \in T$, $\iota(t)$ is a pair $\lambda(t), \gamma(t)$, where:
  - either $\lambda(t)$ is a finite multiset of high-level actions, or $\lambda(t) \in \mathbb{X}$;
  - $\gamma(t)$ is the guard of $t$, which is a Boolean expression on $\text{Var}$ and $\text{Val}$.

We denote by var($t$) (a subset of $\text{Var}$) the set of variables appearing in the annotations of $t$ and its arcs;

- for each arc $(s, t)$, $\iota(s, t)$ is a multiset of structured annotations on $\text{Var} \cup \text{Val}$, representing the values consumed during a firing of $t$; similarly, the values produced during a firing of $t$ are represented by the annotation $\iota(t, s)$;

$N$ is ex-restricted, ex-oriented and $T$-restricted (like for boxes, using $\iota(x, y) = \emptyset$ or $\iota(x, y) \neq \emptyset$ instead of $W(x, y) = 0$ or $W(x, y) > 0$ respectively).

The structured annotations are formalised in [37] and illustrated in the following, see for instance figures 5 and 6. They include, depending on the context, constants and variables, but also more complex terms (introduced later on, possibly including distinguished symbols $\zeta$ and $\varphi$) that encode, for each mode, different sets of values.

In the figures, the hierarchical transitions are represented using double lines (the hierarchical symbols being the capital letters $X$, $Y$, etc.). Entry places are depicted with an incoming double arrow, exit places with an outgoing double arrow and internal places with no double arrow. Also, the notations are often simplified: an empty communication label or an empty (true) guard are generally omitted, arcs with empty annotations are never represented, the singleton multisets are replaced by their unique element, the places are not always named, etc. The purpose of these simplifications is to alleviate the presentation in order to focus on the aspects serving directly the understanding.

We will formalise now different notions allowing us to define the unfolding and the transition rule of $M$-nets. This will allow us to state the property of consistency of the behaviour of an $M$-net with respect to the behaviour of its unfolding.

A binding of a transition $t$ is a substitution $\sigma : \text{var}(t) \rightarrow \text{Val}$. If $x$ is an entity (expression, action, etc.) which depends on the variables in $\text{var}(t)$, we denote by $\sigma(x)$ the evaluation of $x$ under $\sigma$. A transition $t$ with a guard
\( \gamma(t) \) may be fired with a binding \( \sigma \) only if, for all \( g \in \gamma(t) \), \( \sigma(g) \) is true. Moreover, if \( \forall s \in S : \sigma(\iota(s,t)) \in \text{mult}(\alpha(s)) \land \sigma(\iota(t,s)) \in \text{mult}(\alpha(s)) \), i.e., if the annotations on the arcs evaluated under \( \sigma \) respect the types of the places, we say that \( \sigma \) activates \( t \), i.e., that \( \sigma \) is a mode for \( t \).

A marking of an M-net \((S, T, \iota)\) is a function \( M : S \to \text{mult}(\text{Val}) \) which associates to each place \( s \in S \) a multiset of values from \( \alpha(s) \). We distinguish the entry marking for which \( M(s) = \alpha(s) \) if \( \lambda(s) = e \) and \( M(s) = \emptyset \) otherwise, and the exit marking for which \( M(s) = \alpha(s) \) if \( \lambda(s) = x \) and \( M(s) = \emptyset \) otherwise. The typical expected behaviour of an M-net is to start its execution from its entry marking, finishing, if ever, with its exit marking.

The unfolding operation, \( \text{unf} \), associates a low-level labelled net \( \text{unf}(N) \) to any M-net \( N \) and also the marking \( \text{unf}(M) \) of \( \text{unf}(N) \) to any marking \( M \) of \( N \). Examples are given in the figures 1 and 2.

**Definition 5** Let \( N \equiv (S, T, \iota) \) be an M-net. The unfolding of \( N \) is the low-level labelled net \( \text{unf}(N) \equiv (\text{unf}(S), \text{unf}(T), W, \lambda) \) such that:

- \( \text{unf}(S) \equiv \{ s_v | s \in S, v \in \alpha(s) \} \), and for all \( s_v \in \text{unf}(S) \) we have \( \lambda(s_v) = \lambda(s) \);
- \( \text{unf}(T) \equiv \{ t_\sigma | t \in T, \sigma \text{ is a mode for } t \} \), and for all \( t_\sigma \in \text{unf}(T) \) we have \( \lambda(t_\sigma) = \sigma(\lambda(t)) \);
- \( W(s_v, t_\sigma) \equiv \sum_{x \in (s_v,t_\sigma) \land \lambda(x) = \lambda(x)} \iota(s,t)(x) \) and analogously for \( W(t_\sigma, s_v) \).

Let \( M \) be a marking of \( N \). The marking \( \text{unf}(M) \) of \( \text{unf}(N) \) is defined as follows: for all places \( s_v \in \text{unf}(S) \), \( \text{unf}(M)(s_v) \equiv M(s)(v) \). So, the number of (black) tokens in \( s_v \) is the number of occurrences of \( v \) in \( s \).

Such a definition of the unfolding corresponds to the intuition presented in figures 1 and 2: the places and the transitions are unfolded separately, the arcs are derived from the corresponding bindings and the marking of the low-level net reflects directly that of the M-net. We have the following static property:

**Proposition 1** The unfolding of an M-net \( N \) is a box.

The dynamic aspects of the net are captured though to the following definition of the transition rule.

**Definition 6** A transition \( t \) may fire at a marking \( M \) and for a mode \( \sigma \) iff for all places \( s \in S \), we have \( \sigma(\iota(s,t)) \leq M(s) \). The firing of \( t \) produces the action \( \sigma(\lambda(t)) \) and leads to a new marking \( M' \) defined by \( \forall s \in S : M'(s) \equiv M(s) - \sigma(\iota(s,t)) + \sigma(\iota(t,s)) \).

Like for low-level nets, it is possible to consider various kinds of semantics (interleaving, step or truly concurrent). It is also possible to prove the consistency property relating the behaviour of a high-level net with that of its unfolding [8].

**Proposition 2** A transition \( t \) may be fired in the M-net \( N \) at a marking \( M \) for the mode \( \sigma \), leading to the marking \( M' \) iff \( t_\sigma \) may be fired in \( \text{unf}(N) \) at the marking \( \text{unf}(M) \), leading to the marking \( \text{unf}(M') \). The elementary actions corresponding to the executed transition in both nets are the same: \( \sigma(\lambda(t)) = \lambda(t_\sigma) \).
2.3 M-net refinement

One of the essential features of a modular Petri net model is the ability of expressing various levels of abstraction, i.e., the possibility to specify systems in an incremental way using successive refinements. It is also important to be able to define the operations on the model in a simple and homogeneous way. These features can be provided thanks to a transition refinement (substitution), allowing one to replace a transition (describing an abstract, simple, behaviour) by a net (describing a concrete, detailed, behaviour). Approaches like, for instance, [5, 6], propose very general solutions for low-level (box-like) nets. However, for a long time, a similar flexibility had not been obtained for high-level nets. Indeed, the approaches introducing the concept of hierarchy like for instance [46] or [12] in coloured nets, [25] in algebraic nets, or [45] in M-nets, work only in very restricted cases. These results were generalised in [37] which removed most of the previous restrictions.

The substitution of a hierarchical transition $t$ (labelled by $X$) in an M-net $N$ by an M-net $N'$ means that $t$ is replaced in $N$ by a copy of $N'$; this is denoted by $N[X \leftarrow N']$. In order to do this, the transition $t$ is removed and its pre-places are combined with the entry places of $N'$ while its post-places are combined with the exit places of $N'$. In other words, $\ast$ is identified with
while \( t^* \) is identified with \( N'^r \). This is performed in such a way that any execution of \( t \), for any mode, in \( N \) is replaced in \( N[X \leftarrow N'] \) by an execution of \( N' \) from its entry marking to its exit marking. We wish also that all the transitions labelled \( X \) can be replaced simultaneously, which allows the net \( N' \) to be executed (possibly concurrently) for each replaced transition.

The basic mechanism used here (see also [36, 37, 78]) in order to guarantee the consistency with respect to the unfolding of the M-net refinement and that of boxes, is that of labelled trees. It is a generalisation of the Cartesian product used by other authors in order to define combinations of places.

In the case of boxes, the labelled trees are used in order to generate the places combining the pre-places and the post-places of refined transitions with the entry and exit places of the refining nets. The main difference in the case of M-nets is that only one place is generated during each combination, but its type is a set of labelled trees. These have at most two levels, one level corresponds to what comes from the refined net \( N \) and the second level corresponds to what comes from the refining net \( N' \). Labelled trees like that shown in figure 3 are used in order to generate new structured values belonging to the types of interface places in the refined net (i.e., the places which are connected to one or more hierarchical transitions).

![Diagram](image)

**Fig. 3** Illustration of the scheme of the value trees of the interface places.

This M-net refinement allows to overcome most of the restrictions present in previous approaches. In particular, arbitrary types are allowed for the places around hierarchical transitions and for the entry/exit places of refining nets. Moreover, arbitrary arc annotations and side-loops on hierarchical transitions are also admitted.

Let \( N \overset{\text{def}}{=} (S, T, \iota) \) be an M-net, \( X \in X \) be a hierarchical symbol, \((N_i)_{i \in I}\) be a family of M-nets and \((X_i)_{i \in I}\) be a family of hierarchical symbols. We use the following notations:

- \( X_N \overset{\text{def}}{=} \{ X_i \mid i \in I \} \) is the set of hierarchical symbols of \( N \);
- \( T^X \overset{\text{def}}{=} \{ t \in T \mid \lambda(t) = X \} \) is the set of hierarchical transitions labelled by \( X \);
- \( T^{X_i} \overset{\text{def}}{=} \bigcup_{i \in I} T^{X_i} \) is the set of hierarchical transitions of \( N_i \);
- for each set \( R \subseteq S \cup T \), we consider the sets \( R^* \overset{\text{def}}{=} \bigcup_{x \in R} x^* \), \( *R \overset{\text{def}}{=} \bigcup_{x \in R} *x \) of successors and predecessors of the nodes in \( R \);
- \( \bar{N} \overset{\text{def}}{=} N[X_i \leftarrow N_i] \overset{\text{def}}{=} (\bar{S}, \bar{T}, \bar{\iota}) \) is the net \( \bar{N} \) in which each transition labelled by \( X_i \) is replaced by a copy of \( N_i \), for all \( i \in I \). All these substitutions are simultaneous. We call \( N \) the refined net, the \( N_i \)'s the refining nets and \( \bar{N} \) the resulting net.

The places of the resulting net \( \bar{N} \) belong to one of two possible categories:
either they are the places of the refined net \( N \) (some of them are the interface places if they are connected to one or more hierarchical transitions);

- or they are copies of internal places of the refining nets \( N_i \).

This way, each place \( s \) of \( N \) is also a place of the refined net, with the same status. The only difference is in its type which is then the set of labelled trees constructed from the values in the type of \( s \) and from the values coming from the types of entry/exit places of the refining nets \( N_i \). The new type of \( s \), \( \tilde{\alpha}(s) \), is the set of all labelled trees (up to isomorphism) of the form represented in figure 3 where:

- the root is labelled by a value \( v \in \alpha(s) \);
- the edges are labelled by pairs \((t, \sigma)\) where \( t \) is a hierarchical transition of \( N \), adjacent to \( s \) and \( \sigma \) is a mode for \( t \);
- the direction of the edges corresponds to the direction of the arc between \( t \) and \( s \) in \( N \) (down if there is an arc \((s, t)\), up otherwise);
- the leaves are labelled by pairs \((s_i, w_j)\) where \( s_i \) is an entry or exit place of the refining net \( N_i \) and \( w_j \) is a value in the type of \( s_i \).

For example, the new values of an interface place connected as shown in figure 4 in the refined net, are of the form depicted on the right. We assume that the transition \( \sigma_1 \) is a mode for \( t_1 \) that is refined by an M-net \( N_1 \) having a unique entry place \( e \) with the value \( \bullet \) in the type of \( e \). Respectively, \( \sigma_2 \) is a mode for \( t_2 \) that is refined by an M-net \( N_2 \) having a unique exit place \( x \) with the value \( \bullet \) in the type of \( x \). The root of the value is labelled by a value from the original type of \( s \) and the leaves by the values of the places \( e \) and \( x \).

![Fig. 4 An interface place in a refined net and an example of a value tree in the type of this place in the refined net.](image)

More precisely, as illustrated in figure 5, for all \( i \in I \), for all \( t \in s^* \) labelled by \( X_i \), and for all modes \( \sigma \) of \( t \) such that \( v \) belongs to \( \sigma(\iota(s, t)) \), there is in the tree an edge labelled \((t, \sigma)\) going down to a leaf labelled by \((e, w)\) where \( e \in N_i^e \) and \( w \in \alpha(e) \). Symmetrically, for all \( i \in I \), for all \( t' \in s^* \) labelled by \( X_i \), and for all modes \( \sigma' \) of \( t' \) such that \( v \) belongs to \( \sigma'(\iota(s, t'))\), there is an edge labelled \((t', \sigma')\) which goes up from a leaf labelled by \((x, w')\) where \( x \in N_i^x \) and \( w' \in \alpha(x) \). Notice also, that for the places \( s \) in \( N \) which are not connected to any hierarchical transition to be refined (i.e., if \( T^X \cap \{s \cup s^*\} = \emptyset \)) the trees are reduced to their roots and we have \( \tilde{\alpha}(s) \subseteq \alpha(s) \).

The internal places of the refining nets \( N_i \) lead to the set \( S_i \) of internal places of the resulting net, with \( S_i = \{t, s_i \mid t \in T^X \land s_i \in N_i^{s_i}\} \). The status of these places is still internal and their types are composed of values of the
form \( \sigma.v \) (trees reduced to their roots) where \( \sigma \) is a mode for \( t \) and \( v \) is a value from \( \alpha(s_i) \).

The transitions of the resulting net \( \tilde{N} \) may be of two kinds:

- the transitions \( t \in T \setminus T_{N,i}^{X,s} \) from \( N \) which are not substituted and whose inscriptions are unchanged, i.e., \( \tilde{i}(t) \equiv i(t) \);
- the transitions copied from each \( N_i \), belonging to \( T_i \equiv \{ t.t_i \mid t \in T^i \land M_i \in T_i \} \).

The arcs may be of three different kinds:

- those connecting places coming from \( N \) to transitions which are not substituted in \( N \);
- those connecting places coming from \( N \) with transitions coming from \( N_i \);
- and finally, those connecting internal places and transitions both coming from \( N_i \).

These three kinds of arcs are illustrated on the right hand side in figure 6:

- the transition \( t' \) was untouched by the refinement, but the place \( s \) has now a type composed of labelled trees. The firing of \( t' \) has to produce in \( s \) one instance of each tree belonging to the type of \( s \). Each root corresponds to one value produced in \( s \) by the firing of \( t' \) in \( N \). The notation \((x,s),\varphi\) corresponds to this case and the distinguished symbol \( \varphi \) means that the matching leaves of the tree may be arbitrarily labelled;
- the transition \( t.r \) has to consume from \( s \) one instance of each tree whose root is labelled by the value corresponding to the binding of \( x \) during the firing of \( t \) in \( N \). Each tree has a leaf labelled by the value consumed (in \( e_1 \) or \( e_2 \)) via \( z \) during the firing of \( r \) in \( N' \). The notations \((x,s),(z,e_1)\) and \((x,s),(z,e_2)\) correspond to this case;
- finally, the transition \( t.r \) has to produce in \( l.s' \) one instance of each value \( \sigma.v \) where \( v \) is a value produced via \( u \) during the firing of \( r \) in \( N' \) and \( \sigma \) is a mode for \( t \) in \( N \). The notation \( \zeta(u,s') \) corresponds to this case and the
distinguished symbol $\zeta$ denotes that the first part of the token matches any value allowed by the place types.

The formal definition of the refinement, of the evaluation of arcs and also of the transition rule of a refined net follows exactly this intuition. It is presented in the next section in a more general context. It guarantees a number of properties (cf. [37]), for instance:

**Theorem 1** Let $N$, $N_i$, $N'_i$, and $N''_i$ (for $i \in I$, $j \in J$, and $k \in K$) be M-nets. The following properties hold.

- Domain preservation: $N[X_i \leftarrow N_i \mid i \in I]$ is an M-net.
- Consistency with the low-level model via the unfolding: up to isomorphism,
  \[
  \text{unf}(N[X_i \leftarrow N_i \mid i \in I]) = \text{unf}(N)[X_i \leftarrow \text{unf}(N_i) \mid i \in I].
  \]
- Expansion law: if $J \subseteq I$, $I \cap K = \emptyset$ and $\{Y_k \mid k \in K\} \cap \{X_i \mid i \in I\} = \emptyset$, then, up to isomorphism:
  \[
  N[X_i \leftarrow N_i \mid i \in I][X_j \leftarrow N'_j, Y_k \leftarrow N''_k \mid j \in J, k \in K] \\
  = N[X_i \leftarrow N_i[X_j \leftarrow N'_j, Y_h \leftarrow N''_h \mid j \in J, h \in K], Y_k \leftarrow N''_k \mid i \in I, k \in K].
  \]

The expansion law expresses the commutativity of refinements and is a generalisation of the following property for M-nets $N$, $N'$, $N''$ and $N'''$:

\[
N[X \leftarrow N'][X \leftarrow N'', Y \leftarrow N'''] \\
= N[X \leftarrow N'[X \leftarrow N'', Y \leftarrow N'''], Y \leftarrow N'''].
\]

The refinement is very useful in the process of modelling a system in a structured way. It is also essential for a uniform synthesis of the control-flow operators in the algebra of M-nets as we will see in section 4.
3 Extensions of M-nets

3.1 Parameterised substitution

In [39, 53, 64] the syntax of the parallel specification language B(PN)² [10] having a Petri net semantics has been extended by procedures. The mechanism to manage the different instances of a procedure used in [39, 53, 64] requires the usage of "net parameters" in the refinement in order to be able to distinguish between different active instances. Indeed, if in a sequential environment the instances of a procedure are always totally ordered and can be managed by a stack, it is not the case in a parallel environment, where several active instances of a procedure may be generated by concurrent calls and may evolve concurrently. These instances are distinguished by different modes of the same hierarchical transition, as for instance, the modes $\sigma_1 = (x \mapsto 1)$ and $\sigma_2 = (x \mapsto 2)$ of the transition $f$ in figure 7. The procedure body is modelled by the parameterised M-net $N'(id)$, where the parameter id can take values in $\{1, 2\}$ and is used to identify the instances. The result of the refinement is presented on the right of figure 7.

![Diagram](image)

Fig. 7 Parameterised refinement: management of the instances of a procedure $P$ whose body is given by $N'$.

The parameterised refinement [37] introduces a mechanism that allows to identify the variables from the environment of a hierarchical transition (for instance, $x$ in the net $N$ in figure 7) with the corresponding parameters (such as id in $N'(id)$). Moreover, the solution proposed in [37] is not limited to this particular case but introduces a general scheme which works in all the circumstances.

To start with, we introduce parameterised M-nets [37] in which we will use the following extensions:

- $Par$ denotes the set of net parameters, disjoint from $Val$ and $Var$. Each parameter $\psi \in Par$ is assumed to have a non-empty type denoted $set(\psi)$;
- $\Psi$ is a list of parameters of the form $\psi_1, \ldots, \psi_n$, where $\forall i \in \{1, \ldots, n\}$:
  $\psi_i \in Par$ (with $i \neq j \Rightarrow \psi_i \neq \psi_j$). The set of parameters $\{\psi_1, \ldots, \psi_n\}$ associated to $\Psi$ is denoted by $Par(\Psi)$;
- a substitution $\kappa : Par \rightarrow Var \cup Par \cup Val$ is denoted by $(\psi_i \mapsto \nu_1, \ldots, \psi_n \mapsto \nu_n)$; it is elementary if $\nu_i \in set(\psi_i)$, for all $i \in \{1, \ldots, n\}$;
each action $act(\nu_1, \ldots, \nu_{ar(act)})$ has arguments which are values, variables or parameters, i.e., for all $j \in \{1, \ldots, ar(act)\}$, $\nu_j \in Val \cup Var \cup Par(\Psi)$;

- each hierarchical symbol $X \in \mathbb{X}$ has an arity $ar(X)$ representing the number of its arguments. The term $X(\nu_1, \ldots, \nu_{ar(X)})$ represents a hierarchical parameterised action where all the $\nu_j \in Val \cup Var \cup Par(\Psi)$, for $j \in \{1, \ldots, ar(X)\}$, are the arguments of $X$.

**Definition 7** A parameterised M-net $N$ is a quadruple $(S, T, I, \Psi)$ (often denoted by $N(\Psi)$), where $S, T$ and $I$ are as in definition 5 and satisfy the following conditions:

- the net has an arity which is the size of the list $Par(\Psi)$;

- for each transition $t \in T$, $\lambda(t)$ is either a finite multiset of parameterised actions or a hierarchical parameterised action. The parameters which appear in $\lambda(t)$ belong to $Par(\Psi)$;

- the parameters never appear in guards nor in structured annotations of arcs.$^2$

For a given substitution, the dynamic behaviour of such a net is similar to that without the parameters. Also, the semantics of a parameterised M-net [37] does not depend on the names of the parameters nor on the variables in the environment of each transition; they may be consistently renamed at will.

The unfolding of a parameterised M-net is a family of low-level nets (rather than one low-level net as in the case without the parameters) $unf(N) \Rightarrow unf(N, \kappa)$ where $\kappa$ is an elementary substitution such that each choice of values of parameters specified by $\kappa$ leads to a (generally different) low-level net. As for M-nets without parameters, we have the consistency of the behaviour of a parameterised M-net with that of each low-level net corresponding to it [37].

**Proposition 3** A transition $t$ can fire in a parameterised M-net $N$ at a marking $M$ with the binding $\sigma$ and leads to a marking $M'$ iff $t_\sigma$ can fire in all $unf(N, \kappa)$ at $unf(M)$ and leads to the marking $unf(M')$. The parameterised or hierarchical actions produced during this firing are the same in the high-level net $N$ and in each low-level one $unf(N, \kappa)$.

Using the concepts and the notations introduced in the previous section $(S^i, T^i, \alpha(s), \zeta, \varphi, \text{etc.})$, the parameterised refinement is defined as follows:

**Definition 8** Let $N = (S, T, I, \Psi)$ and $N_i = (S_i, T_i, I_i, \Psi_i)$, for $i \in I$, be M-nets. The refinement $N[X_i \leftarrow N_i(\psi_{i,1}, \ldots, \psi_{i,ar(X_i)}) \mid i \in I]$ (with $i \neq j \Rightarrow X_i \neq X_j$) is defined as the M-net $\tilde{N} \overset{\alpha}{=} (\tilde{S}, \tilde{T}, \tilde{I}, \Psi)$, with the same$^3$ set of parameters as $N$, such that:

- $\tilde{S} \overset{\alpha}{=} \bigcup_{i \in I} S^i \cup S$.

$^2$ This last restriction were removed in [27].

$^3$ The $N_i$’s do not introduce new parameters in the refined net, as their parameters are all instantiated by variables, values or parameters of $N$. 

\[ M-nets: \text{a survey.} \]
\[ T \stackrel{\text{def}}{=} \bigcup_{t \in T} (T^i \cup (T \setminus T^X)), \]

\[ \hat{i}(\hat{s}) \defeq \begin{cases} 
\lambda(\hat{s}), \delta(\hat{s}) & \text{if } \hat{s} \in S, \\
\hat{s} & \text{if } \hat{s} = t.s_i \in S^i, \\
\hat{s} & \text{if } \hat{s} = t.s_i \in S^i, \\
\hat{t} & \text{if } \hat{t} = t (T \setminus T^X) \\
\hat{t} & \text{if } \hat{t} = t (T \setminus T^X) \\
\end{cases} \]

\[ \hat{i}(\hat{t}) \defeq \begin{cases} 
\sum_{a \in \lambda(\hat{s})} \psi(\hat{s}, \hat{t}) \in X_i \text{ if } \hat{t} = t.t_i \in T^i, \\
\sum_{a \in \lambda(\hat{s})} \psi(\hat{s}, \hat{t}) \in X_i \text{ if } \hat{t} = t.t_i \in T^i, \\
0 & \text{otherwise}.
\end{cases} \]

The inscriptions \( \hat{i}(\hat{s}, \hat{t}) \) of the arcs in \( \hat{T} \times \hat{S} \) are defined analogously.

The main difference between the parameterised refinement and the non-parameterised one concerns the label of a transition in the refined net. It contains an application of a substitution \( \varphi \) which defines the correspondence between the arguments \( \nu_i, \nu_i.ar(X_i) \) of the hierarchical transition of \( N \) and the parameters \( \psi_i, \psi_i.ar(X_i) \) of \( N_i \). The rest of the definition is exactly the same as explained for the non-parametrisity case and illustrated in figure 5: in particular, the values of the types of the places (labelled trees) and the arc annotations are constructed exactly in the same way.

**Theorem 2** The properties stated for the non-parameterised refinement hold in the parameterised context, in particular:

- Domain preservation;
- Consistency with the low-level domain via the unfolding;\(^4\)
- Expansion law.

### 3.2 Recursion

The recursion \( \mu\{X_i.N_i \mid i \in I\} \) means “replace in \( N \) all the \( X_i \)’s by \( N_i \)’s, and restart on the result ad infinitum”. This operation has been defined first for low-level nets \([4, 5]\), where the recursion \( \mu\{X_i.N_i \mid i \in I\} \) was interpreted as a kind of “limit” of the refinements \( N[X_i \leftarrow N_i] \) where \( N \) is replaced with \( N_i \). This concept is formalised in \([37]\) for the case of M-nets, including the possibility of parameterised recursion. The semantics of the recursion is given by expressing explicitly the form of the result instead of as a limit like in \([4, 5]\).

\(^4\) Remember that the unfolding of a parameterised M-net gives a family of low-level nets.
There are several similarities between the refinement and the recursion, but the latter is more complex. The transitions are here finite sequences \( t.t_1 \ldots t_j \) of arbitrary length (but satisfying some constraints), instead of sequences of length 1 or 2 like \( t \) or \( t.t \) as used in definition 8. Moreover, the types of the places contain values which are labelled trees of arbitrary height, possibly infinite, instead of the trees of height 1 or 2 as in definition 8.

Intuitively, a sequence of transitions \( t.t_1 \ldots t_j \) expresses a sequence of refinements which begins by the hierarchical transition \( t \) in \( N \), continues with the hierarchical transitions \( t_1, \ldots, t_{j-1} \) in M-nets \( N_i, \ldots, N_{i_{j-1}} \), and finishes with a transition \( t_j \) (hierarchical with a label \( X_j \notin X_N \) or non-hierarchical) in \( N_i \). Each sequence of that form leads to a transition in the recursive M-net \( \tilde{N} \).

As for the refinement, the places \( s \) may be of the form \( \pi.s \) and we distinguish two cases:

- the places coming from \( N \), for which \( \pi \) is empty;
- internal places copied from the nets \( N_i \) for which \( \pi = t_1.t_2 \cdots t_{n-1} \) is not empty.

The new values of these places have the general shape shown in figure 8. The root is labelled by \( \sigma_1.\sigma_2 \ldots \sigma_{n-1}.u \) where \( u \) is a value from \( \alpha(s) \) if \( \pi \) is empty or from \( \alpha_i(s) \) if \( \pi = t_1.t_2 \cdots t_{n-1} \) and \( t_{n-1} \) is labelled by \( X_i \). Each \( \sigma_i \) is a mode for the hierarchical transition \( t_i \) corresponding to a given depth of the recursion. The other labels are analogous to that of refinement, their exact description is given in [37].

As for the refinement, the definition of the recursion is complex, but it has the advantage to work in any circumstance. In particular, it allows hierarchical transitions to be connected to entry or exit places of the nets \( N_i \) (or both), which was not the case with previous attempts.

We have the following properties [37]:

**Theorem 3**
1. Domain preservation: if \( N \) and all the \( N_i \)’s, for \( i \in I \), are M-nets, then \( \mu(X_i, N_i \mid i \in I)N \) is also an M-net.
2. Consistency of the recursion with the low-level nets via the unfolding: for \( N \)-nets \( N \) and \( N_i \), for \( i \in I \), and for all choices \( \kappa \) of values for the

![Diagram of value trees in a recursive M-net](image)

*Fig. 8* The general shape of value trees in a recursive M-net.
parameters, we have, up to isomorphism:

\[ \text{unf}(\mu \{ X_i, N_i \mid i \in I \}) N, \kappa) = \mu \{ X_i(\kappa_i), \text{unf}(N_i, \kappa_i) \mid i \in I, \kappa_i \text{ is elementary} \} \text{unf}(N, \kappa). \]

3. Substitution property: for M-nets \( N \) and \( N_i \ (i \in I) \), we have, up to isomorphism:

\[ \mu \{ X_i, N_i \mid i \in I \} N = N[X_i \leftarrow \mu \{ X_j, N_j \mid j \in I \} N_i \mid i \in I]. \]

In particular, the third property generalises the standard fix point equation \( \mu \{ X.N \} N = N[X \leftarrow \mu \{ X.N \} N]. \)

This approach of the recursion produces potentially infinite nets, a finite representation of which was investigated in [26].

3.3 Buffered communication

An extension introduced in [54, 71] allows M-nets to support buffered communication. The key of this extension is a new kind of places, called buffer places, which are automatically merged when several M-nets are combined, i.e., during the refinement. Consider for instance two M-nets \( N_1 \) and \( N_2 \) being refined in a net \( N \); if a transition \( t_1 \) from \( N_1 \) produces a token in a buffer place and if a transition \( t_2 \) from \( N_2 \) consumes a token from its own buffer place, after the refinement, the place merging ensures that these transitions use the same buffer place, making possible the communication between \( t_1 \) and \( t_2 \), even if they are not originated from the same M-net.

More precisely, we consider a set \( \mathbb{B} \) of buffer symbols such that each \( b \in \mathbb{B} \) is associated with a type \( \text{type}(b) \subseteq \text{Val} \). Each buffer place \( s \) is labelled by a \( b \in \mathbb{B} \) and we have \( t(s) \leftarrow b. \text{type}(b) \). Then, the refinement and the recursion (parameterised or not) are redefined as follows: (1) we use the original definition considering the buffer places as internal ones; (2) for all \( b \in \mathbb{B} \), we merge the buffer places sharing the same label \( b \in \mathbb{B} \), their markings being added.

The extension with buffered communication turns out to be very useful and generally allows to express complex systems using smaller and simpler parts. Many applications of M-nets rely on the buffered communication extension. In particular, we will see later on how the semantics of a parallel specification language can be simplified, thanks to buffer places, by using just combinations of basic nets having only one transition.

4 M-net algebra

The model by which M-nets are inspired, the Petri Box Calculus (PBC) [4, 5] and its more recent versions [6, 34], is a process algebra provided with a low-level Petri net semantics. It has a syntactical domain of box expressions and

\footnote{As shown later on, various M-nets composition operations can be defined. Since they are based on the refinement, they do not require any change to support buffered communication.}
a semantical domain of boxes. Each syntactical operator has a corresponding operation (with the same name) in the net domain. The semantics can be obtained by associating a box to a box expression and then through the standard definition of concurrent evolutions (processes) \cite{3, 4} in order to obtain, for instance, a partial order semantics. The semantics can also be obtained directly from a box expression following the rules of the structured operational semantics following the approach from \cite{6}.

4.1 M-expressions

Even though the model of M-nets has been developed first in the semantical domain, it also has a syntactical domain, called the algebra of M-expressions \cite{58}.

\[ E ::= E_N \mid E \downarrow E \mid E \square E \mid E \parallel E \mid [E \ast E \ast E] \mid E[X \leftarrow E] \mid E[f] \]

\[ \mid E \text{ tie } b \mid E \text{ sy } a \mid E \text{ rs } a \mid E \text{ sc } a \]

The syntax of the algebra of M-expressions \( E \) is presented above, where \( a \in A, b \in B \), and \( f \) is a renaming function on actions, hierarchical and buffer symbols. The operators comprise: sequence \( E_1 \downarrow E_2 \) (the execution of \( E_1 \) is followed by that of \( E_2 \)); choice \( E_1 \square E_2 \) (either \( E_1 \) or \( E_2 \) can be executed); parallel composition \( E_1 || E_2 \) (\( E_1 \) and \( E_2 \) can be executed concurrently); iteration \( [E_1 \ast E_2 \ast E_3] \) (\( E_1 \) is executed once (initialisation). \( E_2 \) can be executed an arbitrary number of times (loop), and then is followed by \( E_3 \) (termination)); refinement \( E_1[X \leftarrow E_2] \) (events labelled by hierarchical symbol \( X \) in \( E_1 \) are replaced by \( E_2 \)); renaming \( E_1[f] \) (elements of \( E_1 \) are consistently renamed by \( f \)); buffer restriction \( E_1 \text{ tie } b \) (the buffer place \( b \) and the related communications become private to \( E_1 \)); synchronisation \( E_1 \text{ sy } a \) (all multi-way synchronisations involving the actions \( a \) or \( a^{\sim} \) are made possible); restriction \( E_1 \text{ rs } a \) (events involving \( a \) or \( a^{\sim} \) may no longer be executed) and scoping \( E_1 \text{ sc } a \) (the synchronisation following by the restriction w.r.t. \( a \)).

We assume that any M-net \( N \) has an associated constant expression \( E_N \) in the M-expression domain. Often, this \( N \) is a very simple M-net composed of only one transition, one entry and one exit place, and possibly connected buffer places, as for instance in \cite{18, 53}.

The correspondence between the syntactical domain of M-expressions and that of M-nets is given compositionally through the semantical function \( mnet \). For the base case we define \( mnet(E_N) \triangleq N \), and the semantics of the operators is as follows, where bin stands for any binary operator in \( \{\downarrow, \square, \parallel\} \) and una stands for any unary operator in \( \{\text{sy } a, \text{ rs } a, \text{ sc } a, \text{ tie } b, [f]\} \), the net operations appearing on the right being defined in the next sections:

\[ mnet(E_1 \text{ bin } E_2) \triangleq mnet(E_1) \text{ bin } mnet(E_2) \]
\[ mnet(E_1[X \leftarrow E_2]) \triangleq mnet(E_1)[X \leftarrow mnet(E_2)] \]
\[ mnet(E_1 \text{ una}) \triangleq mnet(E_1) \text{ una} \]
\[ mnet([E_1 \ast E_2 \ast E_3]) \triangleq [mnet(E_1) \ast mnet(E_2) \ast mnet(E_3)] \]
A fragment of the algebra of M-expressions has also been provided with a
structured operational semantics [18, 58]. It was shown that, for a restricted
class of M-expressions (used for the semantics of the parallel specification
language B(PN)$^2$), the semantics obtained using the operational rules from
the initial state of an M-expression was equivalent to the step semantics of
the corresponding M-net.

4.2 Control flow operations

Originally, at the level of boxes as well as at that of M-nets, the control flow
operations, such as sequence, parallel composition, choice or iteration, were
defined through auxiliary operations of net composition, see for instance [4,
8]. The refinement allows to define these operations in a uniform way. One
uses for that operator nets like those shown in figure 9.

**Fig. 9** The operator nets for the definition of the M-net control flow operations.

**Definition 9** For arbitrary M-nets $N_1$, $N_2$ and $N_3$, we define:

\[
\begin{align*}
N_1 \sqcap N_2 & \iff N_1[X_1 \leftarrow N_1, X_2 \leftarrow N_2] & \text{choice} \\
[N_1 * N_2 * N_3] & \iff N_4[X_1 \leftarrow N_1, X_2 \leftarrow N_2, X_3 \leftarrow N_3] & \text{iteration} \\
N_1 \triangledown N_2 & \iff N_4[X_1 \leftarrow N_1, X_2 \leftarrow N_2] & \text{sequence} \\
N_1 \parallel N_2 & \iff N_4[X_1 \leftarrow N_1, X_2 \leftarrow N_2] & \text{parallel}
\end{align*}
\]

4.3 Synchronous communications

The formal definitions of the operations presented in this section may be
found in [8]; we give here an informal presentation.

The synchronisation $\nu y a$ adds new transitions to the net $N$. It may
be seen as a CCS-like synchronisation extended to multi-sets of actions with
arbitrary arity. Intuitively, one can consider the operation of synchronisation
on an M-net as a result of an application, possibly repeatedly, of a certain
number of partial synchronisations between pairs of transitions having in
their labels conjugate actions $a$ and $\widehat{a}$. The repetition of such binary synchronisations with respect to $a$, for all pairs of transitions which contain conjugate actions, and until no new transition can be added, results in the synchronisation of $N$ with respect to $a$. $N\ sym$. For instance, the transitions $t_1$ and $t_2$ in the fragment of the M-net $N$ in figure 10 may synchronise together w.r.t. $start$. The corresponding basic synchronisation generates a new transition $t$ via a renaming of the variables in the surrounding of $t_1$ and $t_2$ and a unification of the arguments of the actions $\text{start}(x)$ and $\text{start}(z)$, e.g., with $\{x \mapsto z\}$. The guard of this new transition is the disjunction of the guards of $t_1$ and $t_2$, and the multiset of synchronous actions is the sum of those of $t_1$ and $t_2$ minus $\text{start}(x)$ and $\text{start}(z)$, both substituted by the unification discovered for the conjugated actions. The result of this binary synchronisation is shown on the right in figure 10.

\begin{figure}[ht]
\centering
\begin{tikzpicture}
  \node (s1) at (0,0) [shape=circle,draw] {\{\}};
  \node (s2) at (1,0) [shape=circle,draw] {\{\}};

  \node (t1) at (-2,0) [shape=circle,draw] {\{1,2,3\}};
  \node (t2) at (-1,0) [shape=circle,draw] {\{1,2,3\}};

  \node (t3) at (2,0) [shape=circle,draw] {\{\}};
  \node (t4) at (3,0) [shape=circle,draw] {\{\}};

  \draw[->] (s1) -- (t1) node [midway, above] {\text{start}(x)};
  \draw[->] (s1) -- (t2) node [midway, above] {\text{start}(z), \text{get}(z)};
  \draw[->] (s2) -- (t3) node [midway, above] {\text{start}(x)};
  \draw[->] (s2) -- (t4) node [midway, above] {\text{get}(x)};

  \draw[->] (t1) -- (s2) node [midway, above] {x \neq y};
  \draw[->] (t2) -- (s1) node [midway, above] {x \neq y};

  \node (x) at (-2.5,0) {\{1,2,3\}};
  \node (y) at (-1.5,0) {\{1,2,3\}};
  \node (z) at (2.5,0) {\{\}};
  \node (w) at (3.5,0) {\{\}};

  \node (tx) at (-2.5,-1) {\ldots};
  \node (ty) at (-1.5,-1) {\ldots};
  \node (tz) at (2.5,-1) {\ldots};
  \node (tw) at (3.5,-1) {\ldots};

\end{tikzpicture}
\caption{A fragment of an M-net $N$ and its synchronisation w.r.t. $\text{start}$ (where $z$ has been substituted with $x$).}
\end{figure}

The restriction of an M-net with respect to $a$ removes from the net all the transitions which contain in their labels actions involving $a$ or $\widehat{a}$. For instance, the result of $(N\ sym \text{start}) rs \text{start}$ removes from the net represented on the right in figure 10 the transitions $t_1$ and $t_2$ (but preserves $t$); it corresponds also to the looping $N\ sc \text{start}$.

The algebraic properties of the restriction are stated by the theorem 4 (equality of nets being considered up to isomorphism). The same properties are obtained in M-nets for the synchronisation (equality of nets being considered up to isomorphism and up to the equivalence which identifies duplicated transitions [8,35]). This theorem is a corollary of a similar one for a version of M-nets using algebraic data types [5,1].

**Theorem 4** For an M-net $N$ and $a_1, a_2 \in A$, we have:

- commutativity of $\text{sy}$: $(N\ sym a_1)\ sym a_2 = (N\ sym a_2)\ sym a_1$
- idempotence of $\text{sy}$: $(N\ sym a_1)\ sym a_1 = N\ sym a_1$
- commutativity of $\text{rs}$: $(N\ rs a_1)\ rs a_2 = (N\ rs a_2)\ rs a_1$
- idempotence of $\text{rs}$: $(N\ rs a_1)\ rs a_1 = N\ rs a_1$

Thanks to the commutativity of $\text{sy}$ and $\text{rs}$, we shall use the extended notations $N\ sym A$, $N\ rs A$ and $N\ sc A$, for $A \subseteq A$. 
4.4 Buffered communication

The buffer restriction of an M-net $N$ w.r.t. $b \in \mathbb{B}$ is the M-net $N \mathbin{\text{tie}} b$ which is obtained from $N$ by replacing $b$ by $i$ in the place $s_i$ having this status. This is to say that the buffer place $b$ is made internal. This is illustrated in figure 11. If $N$ has no such place $s_i$, applying $\text{tie} b$ leaves the net untouched. We then obtain the following results.

**Theorem 5** Let $N$ be an M-net, $a \in \mathcal{A}$ and $b_1, b_2 \in \mathbb{B}$. Then:

- **commutativity of tie**: $(N \mathbin{\text{tie}} b_1) \mathbin{\text{tie}} b_2 = (N \mathbin{\text{tie}} b_2) \mathbin{\text{tie}} b_1$
- **idempotence of tie**: $(N \mathbin{\text{tie}} b_1) \mathbin{\text{tie}} b_1 = N \mathbin{\text{tie}} b_1$
- **commutativity of tie and sy**: $(N \mathbin{\text{tie}} b_1) \mathbin{\text{sy}} a = (N \mathbin{\text{sy}} a) \mathbin{\text{tie}} b_1$

The last result, which holds up to isomorphism, shows the orthogonality between buffered and synchronous communication schemes. Moreover, thanks to the commutativity of tie, we shall use the extended notation $N \mathbin{\text{tie}} \mathcal{B}$, where $\mathcal{B} \subseteq \mathbb{B}$.

![Fig. 11 A fragment of an M-net $N$ with a buffer place $b$ (depicted with a double line) and its version with $b$ removed by the buffer restriction $N \mathbin{\text{tie}} b$.](image)

4.5 Relabelling

The relabelling $N[f]$ of an M-net $N$ is like $N$ where the action symbols, the hierarchical symbols and the buffer names are consistently renamed using the function $f$ which must be a bijection on $\mathcal{A} \cup \mathcal{X} \cup \mathbb{B}$ such that $f(\mathcal{A}) = \mathcal{A}$, $f(\mathcal{X}) = \mathcal{X}$ and $f(\mathbb{B}) = \mathbb{B}$.

4.6 Consistency of the operations

The consistency of the operations of the M-net algebra with their low-level counterparts is crucial. It means that the unfolding of an M-net $N$ obtained from a composition of M-nets $N_i$, $i \in I$, using operations of the M-net algebra, is equivalent to the low-level net obtained by composing the unfoldings of the $N_i$’s using the corresponding operations in the low-level domain. It allows in particular to model a concurrent system in the high-level framework (where it is easier) and to obtain automatically (thanks to the unfolding) its low-level equivalent where properties may be efficiently verified.
**Theorem 6** The operations of the M-net algebra are consistent with the low-level ones through the unfolding.

For the control-flow operations, this is a corollary of the consistency of the refinement. For the synchronisation and the restriction, the consistency with the low-level model is non-trivial to show, see [8]. It was obtained up to an equivalence which identifies duplicated transitions (i.e., having the same label, the same connectivity and same annotations, up to consistent renaming of variables). Consistency of the buffer restriction was shown in [54, 71].

The consistency of the M-nets operations was also studied in a more general context allowing the presence of net parameters not only as arguments of synchronous or hierarchical actions but also in the guards and in the arc inscriptions [27].

5 Applications of M-nets

5.1 Semantics of parallel specification languages

The definition of M-nets was accompanied in [7, 8] by their application to a formal semantics of the parallel specification language B(PN)²: Basic Petri Net Programming Notation [10]. B(PN)² comprises in a simple syntax most traditional concepts of parallel programming including nested parallel composition, iteration, guarded commands, and communication via both handshake and buffered communication channels, as well as shared variables. Originally, B(PN)² incorporated no procedures, but this important feature was added at the M-net level first in [39, 64] and then in [53]. The main difficulty when dealing with procedures consisted in the treatment of their parameters, which may be passed by value or by reference. The approach from [53] was based on M-net refinement and synchronisation operations inheriting in this way all properties of the M-net model, in particular the consistency with the low-level. Another semantics based on parameterised M-expressions inheriting in this way also the consistency with the low-level. Another semantics based on parameterised M-expressions was proposed in [52, 70] exploiting the introduction of the buffered communication and the properties of the M-expression algebra. With the experience acquired with the modelling of B(PN)², M-nets could also be used in [40] for a semantics of SDL: Specification and Description Language [21].

5.2 Semantics of object orientation

The works in [61–63] provided first attempts in expressing object oriented concepts using M-nets. They were improved in [16, 17, 20] by allowing the definitions of classes with their own fields (attributes and methods), single class inheritance, polymorphism and dynamic binding. This led to the definition of an extension of B(PN)², called BOON (Basic Object Oriented Notation), having a syntax inspired from Java and C++, and a fully compositional semantics in terms of M-nets. It may be seen as an alternative to other Petri net based formalisms capable to express object oriented concepts, which generally use more complex net classes. This is the case, for
instance, for Object Petri Nets (OPN) \[59\], which uses net tokens, or for CO-OPN \[13\] and CLOWN \[22\], which use algebraic Petri nets (nets extended with algebraic data types).

5.3 Semantics of mobility

In order to be able to express mobility, dedicated process algebras have been designed, among which one of the most popular is certainly the \(\pi\)-calculus \[67\]. The basic device for expressing mobility in this framework is to pass a reference (or a channel) to a process through a communication, allowing the recipient to use then the new channel for further interactions with other processes.

On the other hand, Petri nets are generally not considered as naturally suitable for expressing processes with changing structure, such as communicating agents which can be dynamically linked to other agents, possibly depending on previous communications. In this respect, the approaches from \[28, 29, 31, 32\] show that M-nets enriched with read arcs can be successfully applied to the modelling of systems with such a mobility feature and thus to emulate the \(\pi\)-calculus. The basic idea is to use buffer places to store the current value of a channel and to update this value during the evolution. The originality of this approach is that it allows to build a Petri net semantics of a mobile system in a compositional way, using some simple composition rules and only a few basic nets.

These ideas were applied in \[14, 15\] to give a high-level net semantics to the security protocol language (SPL) \[23\] inspired from \(\pi\)-calculus-like process algebras. Subsequently, Needham-Schroeder protocol was considered to illustrate how this semantics could be used in order to establish the violation of the authentication property.

M-nets were also used in \[30, 33\] for a semantics of various versions of Klaim language \[24\] implementing the mobility using the features of multiple data tuple spaces distributed over network nodes. This work has been applied in a case study about the modelling and verification of multi-agents systems \[2\].

5.4 Semantics of preemption

An extension of M-nets was proposed in \[55, 57\] in order to provide them with preemption capabilities. The preemption of a process is an interruption of its execution. It is called a suspension when it is temporary and followed by a resuming, while it is called an abortion when the process is killed. This extension was applied in \[56\] to give a semantics of exceptions in B(PN)\(^2\).

Two unary operators were introduced: the first one, \(\pi_a\), allows an M-net to be suspended and the other, \(\pi_a\), to be aborted. It turned out that defining these operators in the algebraic framework of M-nets required to introduce priorities between the transitions. But it was showed that they were used carefully enough to avoid obtaining the expressive power of Turing machines; instead they allowed a logarithmic simplification of the expression of the
preemption operators. A similar compression was observed in Place Charts Nets introducing abortion in Petri nets [48].

The M-nets extended with priorities and preemption operators are called preemtible M-nets (PM-nets). A PM-net is a pair \((N, \rho)\) where \(N\) is an M-net and \(\rho\) is a binary relation over the transitions of \(N\) which is called the priority relation, following the scheme from [11]. Having \((t_1, t_2) \in \rho\) is denoted \(t_1 \prec t_2\) and means that the enabling of \(t_2\) disables \(t_1\), i.e., that \(t_2\) has the priority over \(t_1\). The refinement is then adapted in order to allow a priority over a hierarchical transition \(t_X\), e.g., \(t_X \prec t\), to be propagated to all the transitions of an M-net \(N'\) refining \(t_X\), which ensures that \(t_X. t' \prec t\) holds for all the transitions \(t'\) of \(N'\). This allows to define the suspension operator in a simple way as \(\pi_a(N) \triangleq N_a[X \leftarrow N] \triangleq sleep\) where \(N_a\) is the net shown on the left of figure 12. Thanks to the priority \(t_X \prec t_2\), all the transitions in the net refining \(t_X\) are suspended as long as \(t_2\) remains enabled. The abortion operator is defined in a similar way using the net \(N_a\) given on the right of figure 12. The main difference is that the looping transition \(t_2\) is exploited to remove the tokens in each place of the PM-net on which \(\pi_a\) is applied. This is made possible by adding to these places emptying transitions (holding the action empty) which are synchronised with \(t_2\). Finally, the abortion of \(N\) is obtained thanks to the priority \(t_3 \prec t_2\) in \(N_a\), which ensures that all the tokens are removed before the net \(\pi_a(N)\) can produce its final marking.

![Fig. 12](image)

Another approach to preemption has been proposed in [74], which consists in a framework where a syntactic level (PBC-like) allows to restrict the concurrency to communicating sequential threads of executions. Preemption is here limited to an exception mechanism, but a thread cannot be suspended or aborted from another thread. The simplification of the concurrency scheme and the restricted preemption allows for a more simpler implementation without priorities.

5.5 Semantics of time constrained specifications

The causal time approach is a way to introduce timing features in an otherwise untimed model [38]. In particular, we shall consider coloured Petri nets as in [77], or M-nets as in [19, 54, 71]. The idea behind causal time is to use the expressivity of the model, this amounts to give an explicit representation of clocks in the modelled systems. In the case of Petri nets, it is
possible to introduce counters and a distinguished tick transition whose role
is to simultaneously increment the counters. Thus, they become the timing
reference and can be used as clock-watches by the processes. This is quite
similar to what exists, e.g., in timed automata [1] except that, in this later
case, clocks are real values which are continuously increased by a process
(the passing of time) external to the system. Using the features of M-nets
allows for a definition of a causal clock which acts like a server for the rest
of the system: each tick counter is associated with an identifier which can be
allocated or freed and may be used to set or check the current value of the
counter from any part of the system. Transmitting the counter identifiers is
made easy by using the buffer places.

Using the causal time approach has been further simplified in [72, 74] by
using a PBC-like syntax allowing to specify time constrained system with-
out the need to manage the identifiers of counters. In particular, [74] allows
to define multiple independent clock ticks, each being associated to various
watch or timeout counters allowing respectively to measure or to constrain
the passing of time. This work also avoids to statically define boundaries to
some counters, which was used to model timeouts in the previous approaches.

It was shown in [19] that the causal time approach is highly relevant since
the model checking can be more efficient using a general purpose model-
checker for high-level Petri nets (MARIE [65]) than using well known tools
for timed automata (Kronos [82] and UPPAAL [60]). This case study was
performed on a railroad crossing problem which is presented in the next sec-
tion. A method to use the concurrent semantics of Petri nets with a notion of
time region was proposed in [72], allowing for more efficient model-checking
of causally timed systems. Another approach has been proposed in [75] to
handle counters of ticks using Petri nets equipped with integer variables. A
compact state space construction was provided in order to give a represen-
tation of the reachable states. This construction aggregates in one unique
symbolic state possibly infinitely many concrete states that differ only by
the values of the integer variables.

6 Example

This section presents an example of a specification using M-nets and causal
time in the version proposed in [19, 71]. The system of interest is a railroad
crossing composed of $n_t$ track sections (each being occupied by only one
train) and of a pair of gates which can prevent cars from crossing the tracks
when a train is present.

The trains are independent and at the beginning none is present in the
crossing. Each of them starts far from the railroad crossing; it triggers a
signal $\text{up}$ when it approaches close enough to the gates. From this point,
it reaches the gates after at least $a_m$ and at most $a_M$ time units. Then, it
passes inside the gates during at least $e_m$ and at most $e_M$ time units and
finally leaves the gates triggering a signal $\text{exit}$.

The gates are initially open. They close in at least $g_m$ and at most $g_M$
time units after receiving a signal $\text{down}$. We assume that they require the
same delay for opening after receiving a signal $\text{up}$. It may happen that the
gates receive the signal \textit{down} when they are already going up; in this case also, the time needed in order to close is within the same bounds.

A controller receives the signals from the trains and reacts by sending signals to the gates at least \( e_m \) and at most \( e_M \) time units. It must ensure the \textit{safety property} which states that if a train is present at the crossing, then the gates must be closed. (In this example, we do not address the \textit{availability property}, i.e., the gates are open as much as possible.)

These various parts of the system are modelled using the three M-nets depicted at the top of figure 13. The M-net given at the bottom of this figure is the causal clock \( N_k \). The number \( n_c \equiv 2 n_t + 1 \) of tick counters in clock \( N_k \) depends on the number \( n_t \) of tracks in the system because we use two counters for each train, with the following static allocation (which improves the efficiency of the verification).

The counter 0 is reserved to the controller, and its maximal value is \( \max_0 \equiv e_M \). This counter is reset when a train is approaching (see the transition \( t_{13} \)) and is used in order to ensure that the signal \textit{down} is sent to the gates after at least \( e_m \) ticks (transition \( t_{14} \)). The maximum number of ticks allowed here, \( e_M \), is enforced in the guard of the transition \( t_{20} \) in the clock. The same counter is also used when the last train leaves the crossing (transitions \( t_{15} \) and \( t_{17} \)). Notice that if we have had different constraints in these two cases, we should have used two different counters. (This is not an intrinsic limitation of causal time but rather a limitation of the simple clock we use.)

The counter 1 is reserved to the gates and its maximal value is \( \max_1 \equiv g_M \). It is reset when the gates receive the signal to go down (transition \( t_5 \)) and it ensures that the gates are down after at least \( g_m \) and at most \( g_M \) ticks (see the transition \( t_7 \) and the guard of \( t_{20} \)). The same counter is used in order to ensure the opening of the gates under the same timing constraints.

For each track \( i \), for \( i \in \{1, \ldots, n_t\} \), we use two distinct counters: \( 2 i \) and \( 2 i + 1 \), with \( \max_{2i} \equiv a_M \) and \( \max_{2i+1} \equiv e_M \). When a train approaches, at least \( a_m \) and at most \( a_M \) ticks can occur between the sending of the signal \textit{app} and the arriving of the train between the gates. This constraint is ensured by the counter \( 2 i \) (transitions \( t_{1} \) and \( t_{2} \)). The counter \( 2 i + 1 \) ensures that there must be at least \( e_m \) and at most \( e_M \) ticks between the crossing of the road by a train and its leaving sending the signal \textit{exit} (transitions \( t_{2} \) and \( t_{3} \)). In particular, \( t_2 \) fires when the train enters the crossing; the counter \( 2 i \) must then indicate a value greater than \( a_m \) (thanks to \( t \geq a_m \) in the guard \( t_2 \)) and the counter \( 2 i + 1 \) is set to zero.

The complete system is then specified as the parallel composition of its constituting nets followed by a scoping w.r.t. all the visible actions. In particular, the actions \( i_t, i_c, i_g \) are used to allow a simultaneous initialisation of all the components. So we have:

\[
N_{rc} \equiv (N_t \| N_p \| N_c \| N_k) \; sc \{ t_t, i_g, i_c, \textit{app}, \textit{exit}, \textit{down}, \textit{up}, \textit{clock} \}.
\]

This system can be modelled using the PEP toolkit [41] which allows for editing and composing M-nets. For the model checking of M-nets, PEP relies on the unfolding, which is unfortunately intractable in this case because of the infinite type of the place \textit{Time} in \( N_k \). Bounding the type, is not enough since
Fig. 13 The nets $N_1$, $N_2$, $N_c$, and $N_u$ (from top to bottom, if taken separately) or their parallel composition (if taken as a single net).
it still has many values which do not correspond to actual markings but lead to a huge unfolding, intractable as well. This explains why MARIA [65] was used in [19] instead of PEP. After the publication of [19], the tool PUNF [49, 50] became available and integrated in PEP, allowing to overcome this limitation.

7 Conclusion

We presented the model of M-nets, based on labelled high-level Petri nets and proceed with various composition operations. We shown that these features make it suitable for bottom-up design by using its process algebra like compositions, as well as for top-down design by using refinement allowing to replace transitions by arbitrary M-nets. Thanks to the operation of unfolding, M-nets are consistent with the Petri Box Calculus (PBC). This characteristics offers a strong formal basis and allows for efficient model checking. Indeed, PBC is based on safe (1-bounded) Petri nets for which optimised techniques exist.

The main extensions of M-nets allow to express parameterised refinement, recursion and buffered communication. The main applications are the definitions of semantics for parallel specification languages, the modelling of object oriented, mobile, preemptive or timed systems. A more detailed example is developed for illustrating this last application. It concerns a railroad crossing system and allows to show how M-net modularity may facilitate the modelling of such time constrained critical systems.

Future works about M-nets will aim at improving the process algebraic aspects, trying to introduce most of the high-level features of M-nets in PBC-like algebras, exploiting in particular, the possibilities of abstraction offered by buffered communication (see, e.g., [72, 74] as first steps). Moreover, work is in progress concerning an efficient verification method for M-nets. The main idea consists in avoiding going through the unfolding way; instead, staying at the high-level allows to envisage approaches based on abstraction (like in [75]) and exploiting the modular aspect of the verified systems.

Acknowledgements

The authors would like to warmly thank the anonymous referees who provided extremely detailed reports that were really useful in improving the paper.

References

29. R. Devillers, H. Klaudel and M. Koutny. Petri net semantics of the finite
30. R. Devillers, H. Klaudel and M. Koutny. A Petri net semantics of a sim-
ple process algebra for mobility. EXPRESS/CONCUR’05, ENTCS 151(3).
31. R. Devillers, H. Klaudel and M. Koutny. Petri net semantics of the finite
32. R. Devillers, H. Klaudel and M. Koutny. A Petri net translation of π-calculus
33. R. Devillers, H. Klaudel and M. Koutny. Modelling mobility in high-level
Petri nets. ACSD’07. IEEE, 2007
34. R. Devillers, H. Klaudel, M. Koutny and F. Pommerene. Asynchronous box
35. R. Devillers, H. Klaudel and E. Pelz. An algebraic box calculus. JALC 5(2).
Univ. of Magdeburg, 2000.
36. R. Devillers, H. Klaudel and R.-C. Riemann. General refinement for high
37. R. Devillers, H. Klaudel and R.-C. Riemann. General parameterised refine-
38. R. Durchholz. Causality, time and deadlines. Data & Knowledge Engineer-
39. H. Fleischhacker and B. Grünbaum. A Petri net semantics for B(PN)2 with
40. H. Fleischhacker and B. Grünbaum. A compositional Petri net semantics for
41. B. Grünbaum and E. Best. PEP – More than a Petri net tool. TACAS’96,
42. H. Genrich and H.J. Lautenbach. System modelling with high-level Petri
43. H.J. Genrich, K. Lautenbach and P.S. Thiagarajan. Elements of general net
theory. AGNTTPS’80, LNCS 84. Springer, 1990.
44. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. In-
Riddlesheim, Germany, 1999.
47. K. Jensen. Coloured Petri nets. Basic concepts, analysis methods and prac-
48. M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin and
A. Yakovlev. Coupling asynchrony and interrupts: place chart nets and their
homespages.cs.ncl.ac.uk/victor.khomenko/home.formal/tools/pnuf
50. V. Khomenko and M. Koutny. Branching processes of high-level Petri nets.
51. H. Klaudel. Modèles algébriques, basés sur les réseaux de Petri, pour la só-
53. H. Klaudel. Compositional high-level Petri net semantics of a parallel pro-
55. H. Klaudel and F. Pommerene. A concurrent and compositional Petri net
semantics of preemption. IFM’00, LNCS 1915. Springer, 2000
56. H. Klaudel and F. Pommerene. A concurrent semantics of static exceptions
in a parallel programming language. ICATPN’01, LNCS 2075. Springer, 2001
64. J. Lilis and E. Pelz. An M-net semantics for B(PN)$^2$ with procedures. ISCIS’96, 1. Middle East Technical Univ., 1996.