Modular State Spaces for Prioritised Petri Nets

C. Lakos[1] and L. Petrucci[2]

[1] University of Adelaide
Adelaide, SA 5005
AUSTRALIA
Charles.Lakos@adelaide.edu.au
[2] LIPN, CNRS UMR 7030, Université Paris XIII
99, avenue Jean-Baptiste Clément
F-93430 Villetaneuse, FRANCE
Laure.Petrucci@lipn.univ-paris13.fr

Abstract. Verification of complex systems specification often encounters the so-called state space explosion problem, which prevents exhaustive model-checking in many practical cases. Many techniques have been developed to counter this problem by reducing the state space, either by retaining a smaller number of relevant states, or by using a smart representation. Among the latter, modular state spaces [CP00,LP04] have turned out to be an efficient analysis technique in many cases [Pet05]. When the system uses a priority mechanism (e.g. timed systems [LP07]), there is increased coupling between the modules — preemption between modules can occur, thus disabling local events. This paper shows that the approach is still applicable even when considering dynamic priorities, i.e. priorities depending both on the transition and the current marking.

Keywords: Modular state spaces, prioritised Petri Nets

1 Introduction

State space exploration is a convenient technique for the analysis of concurrent and distributed systems. Its chief disadvantage is the so-called state space explosion problem where the size of the state space can grow exponentially in the size of the system.

One way to alleviate the state space explosion problem is to use modular analysis, which takes advantage of the modular structure of a system specification. The internal activity of the modules is explored independently rather than in an interleaved fashion. Modular state space exploration has yielded significant efficiency gains in the analysis of systems where the modules exhibit strong cohesion and weak coupling [LP04,Pet05]. The benefits arise because the internal activity of individual modules can be explored independently without considering the many possible interleavings of this internal activity. Interaction between modules is only considered at synchronisation (or fused) transitions.

If the system has some form of priority, e.g. time, then the internal activity of the modules is no longer independent. An earlier internal event in one module

will precede a later internal event in another. In this way, a high priority module may preempt *all* activity of a low priority module, even without interaction. If the priority scheme is dense, e.g. real number priorities, then the priorities may eliminate many possible interleavings of activity and modular state space exploration will yield few benefits. However, if the priority scheme is coarse grained, then modular state space exploration may still be of value. This is the situation that we explore in this paper.

We consider Modular Petri Nets which incorporate a dynamic priority scheme similar to that of Bause's work [Bau97]. The scheme is termed *dynamic* because the priority of a transition depends on the current marking, not just on the firing mode. Bause considered the constraints on the priority scheme so that the prioritised net would preserve liveness and home properties of the non-prioritised net, despite having a reduced state space. By contrast, we are interested in the possible benefits of modular state spaces for prioritised nets.

We choose a priority scheme where the greater priority value implies a higher priority. Equally well, we could choose a priority scheme where lower priority values indicate a higher priority. For example, if the priority value was given by an enabling time, earlier timed events would preempt later ones.

The paper is organised as follows. After introducing the basic definitions and notations in section 2, we adapt, in section 3, the modular state space exploration technique from [CP00,LP04] to modular nets with dynamic priorities. Associated algorithms are given in section 4, together with the formal results on which they depend. Section 5 presents experimental results, showing the benefits of the approach. Finally, section 6 summarises the contributions and gives perspectives for future work.

2 Basic Definitions

This section introduces the basic concepts and notations used in the paper. A parallel is drawn between the definitions of Petri nets and prioritised Petri nets, and then between their modular extensions.

2.1 Petri Nets

We first recall the basic definitions and notations for Petri nets:

Definition 1 (Petri nets).
A Petri net *is a tuple* $PN = (P, T, W, M_0)$*, where:*

- P *is a finite set of* places.
- T *is a finite set of* transitions *such that* $T \cap P = \emptyset$.
- W *is the arc* weight function *mapping from* $(P \times T) \cup (T \times P)$ *into* $\mathbb{N}$.
- M_0 *is the* initial marking*, namely a function mapping from* P *into* $\mathbb{N}$.

The elements defining the Petri net behaviour can now be expressed:

Definition 2 (Markings, enabling rule).

- *A* marking *is a function M mapping from P into $\mathbb{N}$. The set of all markings is denoted by $\mathbb{M}$.*
- *A transition $t \in T$ is* enabled *in a marking M, denoted by $M[t\rangle$, iff $\forall p \in P : W(p,t) \leq M(p)$.*
- *When a transition $t \in T$ is enabled in a marking M_1, it may* occur*, changing the marking M_1 to another marking M_2, denoted by $M_1[t\rangle M_2$ and defined by: $\forall p \in P : M_2(p) = (M_1(p) - W(p,t)) + W(t,p)$. The set of markings* reachable *from a marking M is: $[M\rangle = \{M' \mid \exists \sigma \in T^* : M[\sigma\rangle M'\}$ where T^* is the transitive closure of T.*

2.2 Prioritised Petri Nets

We extend the above definitions to prioritised Petri nets, where the priority of transitions is *dynamic*, i.e. it depends on the current marking [Bau97].

Definition 3 (Prioritised Petri net).
A Prioritised Petri net *is a tuple $PPN = (P, T, W, M_0, \rho)$, where:*

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

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

Definition 4 (Prioritised enabling rule).

- *A transition $t \in T$ is* priority enabled *in marking M, denoted by $M[t\rangle^\rho$, iff:*
  - *it is enabled, i.e. $M[t\rangle$, and*
  - *no transition of higher priority is enabled, i.e. $\forall t' : M[t'\rangle \Rightarrow \rho(M,t) \geq \rho(M,t')$.*
- *The definition of the priority function ρ is extended to sets and sequences of transitions (and even markings M):*
  - *$\forall X \subseteq T : \rho(M,X) = \max\{\rho(M,t) \mid t \in X \wedge M[t\rangle\}$*
  - *$\forall \sigma \in T^* : \rho(M,\sigma) = \min\{\rho(M',t') \mid M'[t'\rangle^\rho$ occurs in $M[\sigma\rangle^\rho\}$.*

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

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

underlying Petri Net. With this in mind, the subsequent presentation only includes definitions of prioritised constructs — the non-prioritised versions can be deduced by setting the priority function to a constant zero value.

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

2.3 State Spaces of (Prioritised) Petri Nets

The *state space* (also named *occurrence graph*) of a Petri net is represented as a graph which contains a node for each reachable marking and an arc for each possible transition occurrence. Since state spaces are defined similarly for Petri nets without and with priorities, only the latter definition is given. The sole difference is whether there are priorities or not for the firing rule.

Definition 5 (State space of a prioritised Petri net).
Let $PPN = (P, T, W, M_0, \rho)$*, be a prioritised Petri net. The* Prioritised State Space *of PPN is the directed graph* $PSS = (V, A)$*, where:*

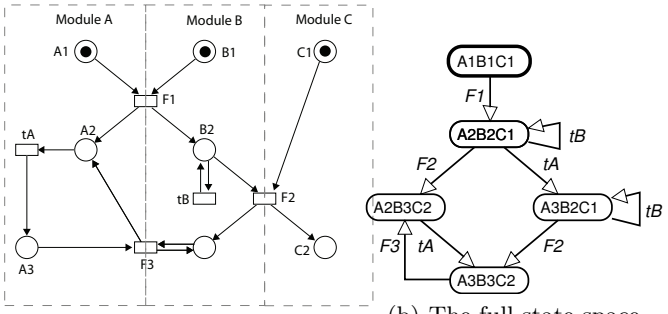
1. $V = [M_0\rangle^\rho$ *is the set of vertices.*
2. $A = \{(M_1, t, M_2) \in V \times T \times V \mid M_1[t\rangle^\rho M_2\}$ *is the set of arcs.*

Example: The Petri net in figure 1(a) is equivalent to the modular Petri net of figure 4. Its (full) state space is shown in figure 1(b). Note that the initial state is shown as A1B1C1, thus indicating that place A1 is marked with a token in module A, place B1 is marked with a token in module B, and place C1 is marked with a token in module C. In this initial state, only transition F1 is enabled, its occurrence leading to state A2B2C1.

Example: The Petri net in figure 2 is a simplified version of the one considered in more detail in figure 5[1]. It captures part of the message-handling of a device, such as those used on a factory floor in the Fieldbus protocol [MSF+99]. The device cycles through states A, B and C. Place U holds one token for each urgent message that is waiting to be sent. At each cycle, the device can send an urgent message (if one is available) by firing transition SU, or it can choose not to send a message by firing transition noS. Similarly, in each cycle, it can generate an urgent message (by firing transition GU), or choose not to generate such a message by firing transition $noGU$.

The state space for this system is shown in figure 3. Here, the states are annotated with the places which hold a token, and place U is flagged with the

[1] The state space for the modular prioritised Petri net of figures 5 and 6 is too large to be represented here.



(a) Petri net equivalent to the modular Petri net of figure 4.

(b) The full state space.

Fig. 1. The Petri net and state space of the system in figure 4.

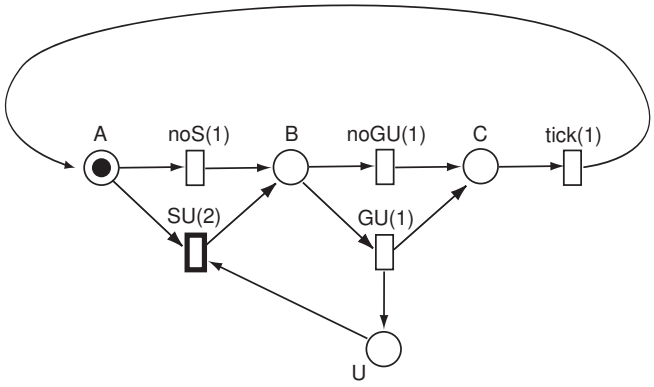


Fig. 2. Simplified Petri net for device message generation.

number of tokens in the place. If the net is *not* prioritised, then the number of urgent messages can grow without limit, as indicated by the incomplete state space. If the transitions are prioritised (with the priorities shown in parentheses), then transition SU has higher priority than $noSU$, and the greyed-out part of the state space will be omitted.

This partial example only illustrates the value of a static priority scheme. The value of a dynamic priority scheme is shown in the extended example of figure 5.

2.4 Modular Petri Nets

Modular Petri nets are defined in a similar manner to Petri nets. Unlike the definitions of [CP00] we only consider communication through transitions.

Definition 6 (Modular Petri Net).
A modular Petri net *is a pair* $MN = (S, TF)$*, satisfying:*

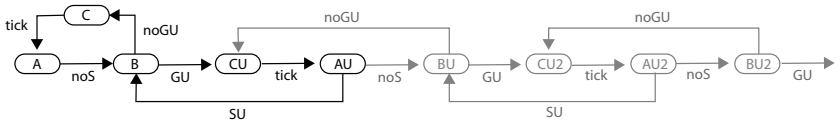


Fig. 3. State space for simplified Petri net for device message generation.

1. *S is a finite set of* modules *such that:*
   - *Each module, $s \in S$, is a Petri net: $s = (P_s, T_s, W_s, M_{0_s})$.*
   - *The sets of nodes corresponding to different modules are pair-wise disjoint: $\forall s_1, s_2 \in S : [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1}) \cap (P_{s_2} \cup T_{s_2}) = \emptyset]$.*
   - *$P = \bigcup_{s \in S} P_s$ and $T = \bigcup_{s \in S} T_s$ are the sets of all places and all transitions of all modules, and $W = \bigcup_{s \in S} W_s$ is the composite weight function defined on all arcs.*
2. *$TF \subseteq 2^T \setminus \{\emptyset\}$ is a finite set of non-empty* transition fusion sets.

In the following, TF also denotes $\cup_{tf \in TF} tf$.
We now introduce transition groups for the actions of the Modular Petri Net, both simple and composite.

Definition 7 (Transition group). *A* transition group *$tg \subseteq T$ consists of either a single non-fused transition $t \in T \setminus TF$ or all members of a transition fusion set $tf \in TF$. The set of transition groups is denoted by TG. The transition groups which consist only of transitions in a set $T' \subseteq T$ is denoted $TG|_{T'}$.*

A transition can be a member of several transition groups as it can be synchronised with different transitions (a sub-action of several more complex actions). Hence, a transition group corresponds to a synchronised action. Note that all transition groups have at least one element.

Next, we extend the arc weight function W to transition groups:

$$\forall p \in P, \forall tg \in TG : W(p, tg) = \sum_{t \in tg} W(p, t),\ W(tg, p) = \sum_{t \in tg} W(t, p).$$

Markings of modular Petri nets are defined as markings of Petri nets, over the set P of all places. The restriction of a marking M to a module s is denoted by M_s.

The enabling and occurrence rules of a modular Petri net can now be expressed.

Definition 8 (Modular Petri net firing rule).

- *A transition group tg is* enabled *in a marking M, denoted by $M[tg\rangle$, iff $\forall p \in P : W(p, tg) \leq M(p)$.*

– *When a transition group tg is enabled in a marking M_1, it may* occur*, changing the marking M_1 to another marking M_2, defined by* $\forall p \in P : M_2(p) = (M_1(p) - W(p, tg)) + W(tg, p)$.

Example: Figure 4 depicts a modular Petri net consisting of three modules A, B and C. Modules A and B both contain transitions labelled F1 and F3, while modules B and C both contain transition F2. These matched transitions are assumed to form three transition fusion sets.

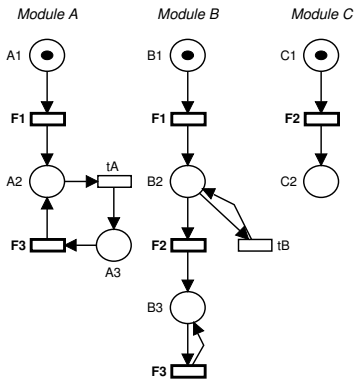


Fig. 4. Modular PT-net with modules A, B and C.

2.5 Prioritised Modular Petri Nets

Similar to modular Petri nets, prioritised modular Petri nets can now be defined:

Definition 9 (Prioritised Modular Petri Net).
A Prioritised Modular Petri net *is a tuple $PMN = (S, TF, \rho)$, where:*

– *(S, TF) is a Modular Petri net.*
– *ρ is the* priority function *mapping a marking and a transition into $\mathbb{R}^+$.*

Transition groups are defined as for non-prioritised Petri nets. They also have an associated priority function:

Definition 10 (Priority of transition groups). *The priority function ρ is extended to transition groups by defining it to be the* minimum *priority of its elements, i.e. $\rho(M, tg) = \min_{t \in tg} \rho(M, t)$.*

Note that the definition of the priority of a transition group is somewhat arbitrary. A simpler approach would have been to insist that all elements of a transition group have the same priority. This would mean that the priority allocations in one module would need to take account of the priority allocations in

all other modules with which this one might synchronise. This seems excessively onerous in practical applications, especially since the priorities must agree for all reachable markings. The decision to define the priority of a transition group as the minimum over the elements has been guided by timed systems — if one transition in a group is enabled earlier than the others, then it must wait till the others are also enabled.

The arc weight function W is extended to transition groups, and markings are defined as for modular Petri nets. The firing rule in prioritised modular Petri nets takes into account the priority of transition groups.

Definition 11 (Priority enabling of transition groups). *A transition group $tg \in TG$ is* priority enabled *in a marking M, denoted by $M[tg\rangle^{\rho}$ iff:*

- *it is enabled, i.e. $M[tg\rangle$, and*
- *no transition group of higher priority is enabled, i.e. $\forall tg' \in TG : M[tg'\rangle \Rightarrow \rho(M, tg) \geq \rho(M, tg')$.*

Example: Figure 5 depicts a module of a prioritised modular Petri net. It captures the message-handling of a device, such as those used on a factory floor in the Fieldbus protocol [MSF$^+$99]. Messages are generated by the device — urgent messages (indicated by U) need to be processed in a timely manner, while normal messages (indicated by N) can wait, but not too long. The device cycles through states A, B, C and D. Place U holds one token for each urgent message that is waiting to be sent, with a maximum of 2 (to limit the size of the state space). The capacity can be imposed by a capacity constraint, or by the use of a complementary place. If place N is non-empty, then there is a normal message to be sent, and the number of tokens indicates how long it has been delayed — it is incremented each time transition *tick* is fired. (This is shown as a double-ended, dashed arc between transition *tick* and place N. The precise notation is not shown to avoid clutter and because it will depend on the specific kind of Petri net.) Again we impose an arbitrary capacity of 2 on this place.

Transition *noS* is local and indicates that no message is to be sent, while transitions SU and SN are fused to others in the environment (as indicated by the bold outlines) and denote that an urgent or a normal message is sent to some controller module. Transition *noGU* indicates that no urgent message is generated in this cycle, while transition GU indicates that an urgent message *is* generated. Similarly, transitions *noGN* and GN relate to the generation of normal messages. (Note that the diagram is again simplified to avoid clutter.)

We attach a priority of 1 to all local transitions. The priority of transition SU is set to the number of pending urgent messages plus 1, while the priority of transition SN is set to the number of ticks that the normal message has been waiting. Thus, if an urgent message is waiting and no normal messages, then transition SU has priority over *noS*. However, if the controller is not ready to receive the urgent message, then transition *noS* will fire. Similarly, if there is no urgent message but there is a pending normal message, then that message will be sent (by firing transition SN) if it has been waiting for more than 1 cycle,

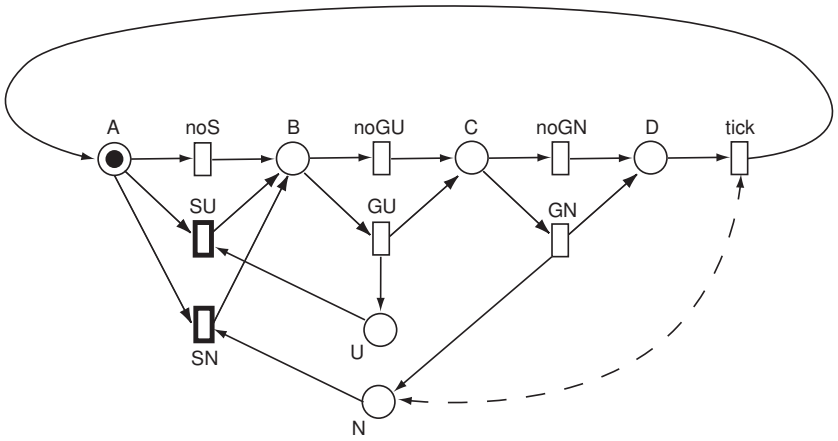


Fig. 5. Module of a prioritised PT-net for the message-handling of a device.

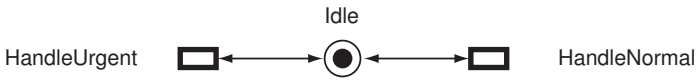


Fig. 6. Module of a controller to receive messages from the devices.

or it *may* be sent if it has been waiting only 1 cycle. Again, a normal message which has been waiting at least 2 cycles, may compete for processing with an urgent message which has only just been generated.

Figure 6 depicts a trivial controller for accepting the messages from a device. It has one transition to handle each of the urgent and normal messages, and we may assume that these transitions have the same priority.

3 Modular State Spaces

In the definition of modular state spaces, we denote the set of states reachable from M by occurrences of local (non-fused) transitions only, in all the individual modules, by $[[M\rangle$.

The notation with a subscript s means the restriction to module s, e.g. $[M\rangle_s$ is the set of all nodes reachable from global marking M by occurrences of transitions in module s only (excluding fused transitions). We will also use lower case m to refer to the local marking of a module.

We use $M_1[[\sigma\rangle\rangle M_2$ to denote that M_2 is reachable from M_1 by a sequence $\sigma \in (T \setminus TF)^* TF$ of internal transitions, followed by a fused transition, i.e. $\sigma = \sigma' tf$ and $M_1[[\sigma'\rangle M_1'[tf\rangle$.

The definition of a modular state space consists of two parts: the state spaces of the individual modules and the synchronisation graph. We can now present

the definition of modular state spaces for prioritised modular Petri nets. The definition of a modular state space for modular Petri nets given in [CP00] uses strongly connected components for optimisation and efficiency purposes. However, the computational benefits of using local strongly connected components are negated by the need for local activity to abide by the global priority function. Hence, strongly connected components are not used in prioritised modular state spaces. Therefore, we will avoid cluttering the paper with the definition of modular state spaces (see [CP00,LP04]), and will directly formulate the prioritised version.

In order to be able to focus on the local context of an individual module, we need to have a localised priority function which is consistent with the global priority function.

Definition 12 (Consistency and locality of priority functions). *Let $PMN = (S, TF, \rho)$ be a prioritised modular Petri net.*

- *The priority function ρ is* consistent *iff $\forall s \in S : \forall t \in T_s : \forall M, M' : M_s = M'_s \Rightarrow \rho(M,t) = \rho(M',t)$.*
- *Given a consistent priority function ρ, we define* local priority functions *ρ_s as the projection onto the local marking, i.e. $\forall t \in T_s, M : \rho_s(M_s, t) = \rho(M,t)$.*

Thus, with a consistent priority function, the priority of a local transition is determined solely by the local marking. If this were not the case, the modularity of the system would be seriously flawed, and the local state space could not be explored without reference to the global state of the system. If it were desired for a local transition to have a priority depending on some global state, then that transition ought to be synchronised with another transition having access to that state.

We can now define the modular state space for prioritised modular Petri nets.

Definition 13 (Prioritised modular state space). *Let $PMN = (S, TF, \rho)$ be a Prioritised Modular Petri net with the initial marking M_0. The* prioritised modular state space *of PMN is a pair $PMSS = ((PSS_s)_{s \in S}, PSG)$, where:*

1. *$PSS_s = (V_s, A_s)$ is the* prioritised local state space *of module s:*
   (a) $V_s = \bigcup_{v \in (V_{PSG})_s} [v\rangle_s^{\rho_s}$.
   (b) $A_s = \{(M_1, t, M_2) \in V_s \times (T \setminus TF)_s \times V_s \mid M_1[t\rangle^{\rho_s} M_2\}$.
2. *$PSG = (V_{PSG}, A_{PSG})$ is the* prioritised synchronisation graph *of PMN:*
   (a) $V_{PSG} = [[M_0\rangle\rangle^\rho \cup \{M_0\}$.
   (b) $A_{PSG} = \{(M_1, (M'_1, tf), M_2) \in V_{SG} \times ([M_0\rangle^\rho \times TF) \times V_{SG} \mid M'_1 \in [[M_1\rangle^\rho \wedge M'_1[tf\rangle^\rho M_2\}$.

Explanation:
(1) The definition of the state space graphs of the modules is a generalisation of the usual definition of state spaces.

(1a) The set of nodes of the state space graph of a module contains all states locally reachable from any node of the synchronisation graph.

(1b) Likewise, the arcs of the state space graph of a module correspond to all priority enabled internal transitions of the module.
(2) Each node of the synchronisation graph is a representative for all the nodes reachable from M by occurrences of local transitions only, i.e. $[[M\rangle^{\rho}$. The synchronisation graph contains the information on the nodes reachable by occurrences of fused transitions.

(2a) The nodes of the synchronisation graph represent all markings reachable from another marking by a sequence of internal transitions followed by a fused transition. The initial node is also represented.

(2b) The arcs of the synchronisation graph represent all occurrences of fused transitions.

The state space graphs of the modules only contain local information, i.e. the markings of the module and the arcs corresponding to local transitions but not the arcs corresponding to fused transitions. All the information concerning these is stored in the synchronisation graph.

It is important to note that the above definition, in contrast to [CP00], introduces a disconnect between the local state spaces and the synchronisation graph. It is not immediately apparent how the computation of the local state spaces in Def. 13 part 1, and specifically of $[v\rangle_s^{\rho_s}$, is used to compute the synchronisation graph in Def. 13 part 2, and specifically $[[M_1\rangle^{\rho}$. This is a significant algorithmic issue which is addressed Section 4.

Example: The modular state space for the modular PT-net of figure 4 is shown in figure 7[2]. Note that there is a local state space for each module, as well as a synchronisation graph which captures the occurrence of fused transitions.

In [Pet05], several experiments were conducted, showing that the size of the modular state space is significantly reduced (compared to the size of the flat state space) when the modules exhibit strong cohesion and weak coupling.

The efficiency gains achievable from modular state spaces arise from the ability to explore local state spaces (of modules) independently, and then combine them via the synchronisation graph to form a composite state space. If desired, a full unfolded state space can be generated from the modular state space (as in Def. 14), though it is computationally more efficient to analyse the system without enumerating all possible interleavings.

[2] The example being a modular Petri net without priorities, the definition of modular state spaces (with strongly connected components) from [CP00] should be used. However, since in that particular case, strongly connected components always contain a single node, the modular state space is also obtained using Def. 13 without considering priorities.

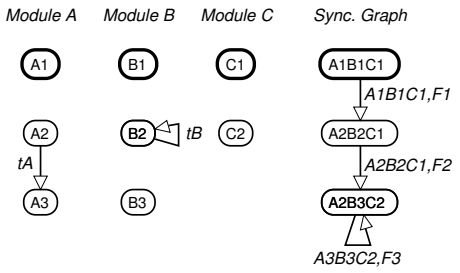


Fig. 7. The modular state space of the system in figure 4.

Definition 14 (Unfolded state space). *Given a prioritised modular Petri net* $PMN = (S, TF, \rho)$ *and its modular state space* $PMSS = ((PSS_s)_{s \in S}, PSG)$, *then the* unfolded state space *of PMSS is* $SS = (V, A)$ *where:*

1. $V = \bigcup_{v \in V_{PSG}} [[v\rangle^{\rho}$.
2. $A = \bigcup_{(v,(m,tf),v') \in A_{PSG}} \{(m, tf, v')\} \cup$
$\bigcup_{m \in V, s \in S, (m_s, t, m'_s) \in A_s, \rho(m,t) = \rho(m, TG)} \{(m, t, (m + m'^{*}_s) - m^{*}_s)\}$,
where $m^{*}_s(p) = m_s(p)$ *for* $p \in P_s$ *and* $m^{*}_s(p) = 0$ *for* $p \in P \setminus P_s$.

The above definition is similar to that of [CP00], except for the addition of priorities. Specifically, part 1 considers states reachable from v by transitions respecting the global priority function, and part 2 considers individual transitions satisfying the same constraint, captured as $\rho(m, t) = \rho(m, TG)$.

The theorem which states the equivalence of the above unfolded state space and the state space of the equivalent non-modular Petri net carries over with only minor changes since both state spaces reflect the priority scheme.

4 Algorithms

In general, modular state spaces can alleviate the state space explosion provided it is possible to construct the modular state space and determine properties based on this state space without needing to explore the possible interleavings of activity between multiple modules, i.e. without having to generate the unfolded state space of Def. 14.

With prioritised modular nets, this is less straightforward because the priority function imposes a global constraint on the behaviour of individual modules. The use of prioritised modular state spaces to determine system properties is the subject of further work. Here, we consider the construction of these prioritised modular state spaces.

The definitions of section 3 are consistent with the definitions of [CP00] but they hide a key computational issue — it is assumed that the computation of

$[[M_1\rangle^\rho$ in Def. 13 part 2 is supported by the computation of the local state spaces in part 1. (A similar comment applies to the computation of $[[v\rangle^\rho$ in Def. 14 part 1.) In other words, the computation of local state spaces is assumed to help determine the global markings reachable from a synchronisation node by the firing of non-fused transitions alone. In the case of non-prioritised modular nets, this is straightforward — localised transition sequences from a synchronisation node can be interleaved in any order. With prioritised modular nets, the interleaving is constrained by the priority function. Accordingly, we need to know whether the local state space, computed with the localised priority function ρ_s, contains *all* the information necessary to compute the interleaved sequences, and then we need to know how to compute such priority-respecting interleaved sequences in an efficient manner. These two questions are addressed in Lemma 1 and Proposition 1, respectively.

Lemma 1. *Given a prioritised modular Petri net* $PMN = (S, TF, \rho)$ *with a consistent priority function* ρ, $M[[\sigma\rangle^\rho M'$ *implies* $M_s[\sigma_s\rangle_s^{\rho_s} M'_s$ *for all* $s \in S$, *where* σ_s *is the restriction of* σ *to the internal transitions of module* s.

Proof. If one ignores the priorities, then it clearly follows that $M[[\sigma\rangle M'$ implies $M_s[\sigma_s\rangle_s M'_s$ forall $s \in S$. In other words, we can split the composite sequence into subsequences for each module. Now, if the priorities are taken into account, then the only way that the result would not hold is that one of the local sequences includes a transition which is not (locally) of maximum priority. But, if it is of maximum priority in the global sequence, then it must be of maximum priority in the local sequence (without fused transitions), in view of the fact that ρ is consistent. □

A consequence of the lemma is that any priority-respecting transition sequence formed from non-fused transitions has its counterpart in local priority-respecting transition sequences of the individual modules. In other words, the local state spaces of Def. 13 part 1 contain all the information necessary to compute $[[M_1\rangle^\rho$ in Def. 13 part 2, and sometimes even more information.

It is important to note that the above lemma means that transitions in local state spaces are provisional, in the sense that they will not necessarily appear in the unfolded state space. This is because their enabling in the unfolded state space depends on the priorities of transitions in other modules. On the other hand, transitions in the synchronisation graph do carry over into the unfolded state space, because these already consider global conditions.

Example: The (abbreviated) state space of Fig. 8 captures both the local state space for the *Device* module of Fig. 5 and the unfolded state space for the system consisting of the *Device* and *Controller* modules of Figs. 5 and 6. The states are encoded as a letter (to indicate which of the places A to D is marked), followed by the number of tokens in place U (which is the number of pending urgent messages), followed by the number of tokens in place N (which is the number of ticks that a normal message, if any, has been waiting). For example, state *C12* means that place C is marked, there is one urgent message pending, and there

is a normal message that has been waiting for at least 2 ticks. Note that dashed arcs indicate that the state is dealt with elsewhere (to avoid many crossing arcs).

For the local state space of the *Device* module, the synchronised transitions *SU* and *SN* of Fig. 8 would not be included — they would only appear in the synchronisation graph. Further, based solely on local information, we cannot be sure whether the transitions shown with dotted arcs will be preempted or not. Thus, in state *A01*, the sending of the normal message competes at the same priority as the transition *not* to send a message. On the other hand, in state *A02*, transition *SN* is of higher priority, but its enabling depends on the enabling of the fusion partners, and hence the alternative *noS* needs to be included as well.

The fusion of the *Controller* net with the *Device* net leads to the same net structure except for the addition of the place *Idle*. Hence the unfolded state space of the composite system is as shown in Fig. 8, except that the dotted arcs (with italic annotations) are omitted because global knowledge of the priorities allows us to deduce that these transitions are preempted by others of higher priority.

Lemma 1 showed that the local state spaces of Def. 13 part 1 capture all the behaviour required to compute $[[M\rangle^{\rho}$ of part 2. We now identify a property that allows it to be computed in an efficient manner.

We first introduce some auxiliary terminology:

Definition 15. *Given a prioritised modular Petri net* $PMN = (S, TF, \rho)$*, and given a local execution sequence* $m[\sigma\rangle_s^{\rho_s} m'$ *in module* s*,* m' *is a* synchronisation point *if* m' *priority enables the local component of a fused transition, i.e.* $\exists f \in T_s \cap TF : m'[f\rangle_s^{\rho_s}$*.* Intermediate synchronisation points *are (potential) synchronisation points in* σ *prior to* m'*. A* realised synchronisation point *is one that is matched by appropriate synchronisation partners.*

The above definition may be clarified by the example of Fig. 9. Having arrived at local marking m' (which is part of a global marking M'), we may find both a local transition t and a transition f, part of a fused transition tf, enabled. If $\rho_s(m', f) < \rho_s(m', t)$, then f cannot be priority enabled whatever the situation with the synchronisation partners. (Recall Def. 10 where the priority of a transition group is the minimum of the priorities of the constituent transitions.) Alternatively, if $\rho_s(m', f) \geq \rho_s(m', t)$, then f is priority enabled locally but tf may not be priority enabled globally. This may be because the synchronisation partners do not enable their component of the fused transition at the same time, or because the priority of the transition group is decreased so that $\rho(M', tf) < \rho_s(m', t)$.

Proposition 1. *Given a prioritised modular Petri net* $PMN = (S, TF, \rho)$*, and given local execution sequences* σ_1*,* σ_2*, ... such that* $M_1[\sigma_1\rangle_1^{\rho_1} M_1'$*,* $M_2[\sigma_2\rangle_2^{\rho_2} M_2'$*, ... then there is a composite execution sequence* σ *with* $M[\sigma\rangle^{\rho} M'$ *if* $\exists s \in S : \rho_s(M_s, \sigma_s) = \min\{\rho_1(M_1, \sigma_1), \rho_2(M_2, \sigma_2), ...\} \geq \rho(M', TG|_{T \setminus T_s})$*, provided no preempting intermediate synchronisation points of* σ_1*,* σ_2*, ... are realised in* σ*.*

Proof. The local sequences can be combined using a simple merge algorithm — at each step, the transition at the head of the local sequence with highest

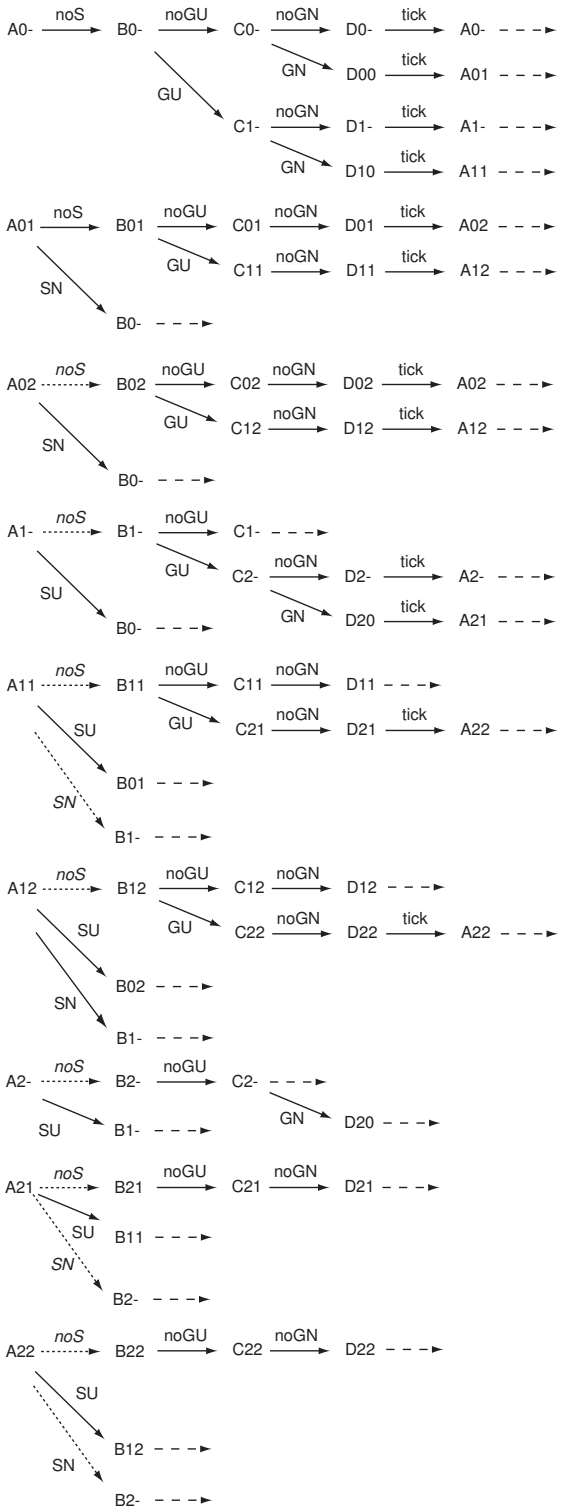


Fig. 8. Local state space of a prioritised PT-net.

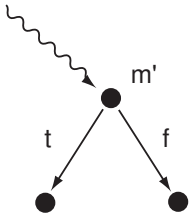


Fig. 9. A synchronisation point.

priority is removed and added to the composite sequence. The choice is only non-deterministic when multiple local sequences have, as head, transitions with the same maximum priority. In this case, choose any one *not* in module s. Note that the non-deterministic choice only affects the order of interleaving and not the final reached marking.

Recall from Def. 3 that the priority of a sequence is the minimum priority of its constituent transitions. So, if the marking M' has priority less than or equal to the priority of all the local sequences, then the transitions of *every* local sequence will be added to the composite sequence *before* any transition enabled in marking M'. If this is not the case, we can still allow a module to have a sequence with minimum priority less than $\rho(M', TG)$, provided that that sequence enables the relevant transitions in marking M', i.e. those with priority $\rho(M', TG)$. □

Corollary 1. *In determining whether a fused transition is enabled, we only need to know the priority of the sequences leading to the synchronisation point and not the sequences themselves.*

We are normally interested in merging local sequences only when it comes time to consider whether a fused transition is enabled. The above proposition and corollary mean that we need only consider the priority of the local sequences leading from one synchronisation point to the next, together with the priorities of the associated sequences.

Further, if we have a priority scheme where the priorities are non-increasing, then the priority of a sequence is given by the priority of the last element of the sequence.

The above formal results provide the foundation for the following algorithms to compute the modular state space. These algorithms are refined versions of those presented in [LP07] for Modular Timed Petri Nets.

Algorithm 1 computes the synchronisation graph, while algorithm 2 computes the local state space for module i. Both follow the common pattern of maintaining a set of as-yet unexplored markings, called `Waiting`. Each iteration of the main loop explores the transitions enabled in these markings, adds new arcs to the relevant state space with the function Arc.Add(...), and adds new nodes to the state space and to `Waiting` with the function Node.Add(...).

In algorithm 2 the local markings are stored together with their preceding synchronisation point — the notation $M_i'[\rangle^p M_i''$ represents local marking M_i''

which is reached from a preceding synchronisation point M_i' with a local transition sequence of priority p. For consistency, a zero length sequence has priority ∞. Lines 8–11 consider the local components of fused transitions — if they are (locally) enabled, then they are added to the eventual result $trysynch_i$ and the marking is identified as a synchronisation point. Lines 12–20 then consider internal transitions. The first alternative deals with the situation where the transition enabling is *not* dependent on the realisation of a synchronisation point, while in the second alternative, the enabling is dependent on such a realisation, and hence the new marking is paired with this synchronisation point. The result of a call to $\textsc{Explore}(S_i, M_i)$ is a set of candidate synchronisations which record the preceding synchronisation point and the priority of the transition sequence leading from one to the other.

In algorithm 1, the central loop (in lines 8–22) tries to match up the candidate synchronisations so that they satisfy the condition of proposition 1. It considers a subset of the elements returned by each call to $\textsc{Explore}(S_i, M_i)$, treating them as a composite sequence from local marking M_i to local marking M_i' with all intermediate synchronisation points identified. In detail:

- Line 9 identifies one of the synchronisation participants — as we shall see below, it is the one with minimum priority sequence.
- Line 10 considers the sequences of local transitions for all modules. In an abuse of terminology, a set of abstract edges from the module is concatenated together to form a sequence, which we then refer to as σ_j which technically should be the sequence of transitions (without the intermediate markings).
- Lines 11–12 consider modules participating in the synchronisation — the end of the returned sequence enables *tf* locally, the priority of the sequence is greater than that from module i, and at the end of the sequence, no local transition will (necessarily) preempt the firing of *tf*.
- Line 13 considers modules *not* participating in the synchronisation — they reach an end point which can lead locally to a marking compatible with the synchronisation, i.e. where the module will wait for the synchronisation.
- Line 14 requires that the minimum priority of module i is greater than the other priorities at the synchronisation point, i.e. so that module i can catch up.
- Line 15 requires that no intermediate synchronisation points are realised, the intermediate synchronisation points being the markings identified in the sequences at line 10. We can determine whether any of these intermediate synchronisation points are realised by applying the same logic as above.

5 Results

The *Maria* tool [Mäk02] was extended with dynamic priorities along the lines of the algorithms in Section 4. Here, we consider some of the results produced with this prototype implementation. The results were produced on a Mac Pro

Algorithm 1: prioritised synchronisation graph.

```
1  set Waiting ← ∅;
2  NODE.ADD(M_0, ∞);
3  repeat
4    forall (M, p) ∈ Waiting do
5      Waiting ← Waiting \{(M, p)};
6      ∀i : trysynch_i ← EXPLORE(S_i, M_i);
7      forall tf ∈ TF do
8        forall M' s.t.
9        ∃i : (tf ∩ T_i ≠ ∅ ∧
10             ∀j : (σ_j = M_j[⟩^{q_j1} M_j1[⟩^{q_j2} ...[⟩^{q_jn} M_jn_j ⊆ trysynch_j ∧
11                  ((tf ∩ T_j ≠ ∅ ∧ M_jn_j = M'_j ∧ M'_j[tf⟩ ∧
12                    ρ_j(σ_j) ≥ ρ_i(σ_i) ∧ ρ(M', tf) ≥ ρ_j(M'_j, T_j \ TF)) ∨
13                   (tf ∩ T_j = ∅ ∧ M'_j ∈ [M_{j,n_j}⟩_j ∧ ρ_j(M'_j) ≤ ρ_i(σ_i) ≤ ρ_j(σ_j))) ∧
14                  ρ_i(σ_i) ≥ ρ(M', TG|_{T\T_i}) ∧
15                  no preempting intermediate synch points are realised))
16       do
17         if M'[tf⟩M'' ∧ ρ(M', tf) = ρ(M', TF) then
18           NODE.ADD(M'', min(p, ρ(M', tf)));
19           ARC.ADD(M[(M', tf)⟩^{ρ(M', tf)} M'');
20         endif
21       endfall
22     endfall
23   endfall
24 until stable ;
```

with two 2.66 GHz dual-core Intel Xeon processors and 2 GB memory. Note that this is not a complete implementation but it is sufficient to provide a proof of concept.

A simple example of message-handling for devices, such as those used on the factory floor in the Fieldbus protocol [MSF+99], was introduced in Section 2. The state space sizes and machine resource requirements are shown in Table 1.

On the left are the results for the modular state space for between 1 and 5 devices. The number of nodes and arcs are the figures for the synchronisation graph, while the time and space requirements (in seconds and kilobytes) are for the construction of the entire modular state space. The size of the state space for each module is 39 nodes and 48 arcs. This is similar to the flat state space for 1 device. Note, however, that the local state space does *not* include the fused transitions, while the flat state space will record their occurrence.

On the right of the table are the results for a flattened system, i.e. the unfolded state space. Note that memory was exhausted for 5 devices.

It might be argued that if the devices were identical, then symmetry reduction would probably give similar, if not better, results. However, if the devices were *not* identical, then the modular state space exploration would still be effective.

Algorithm 2: prioritised local state space — EXPLORE(S_i, M_i).

```
1  set Waiting_i ← ∅;
2  set trysynch_i ← ∅;
3  NODE.ADD(M_i[⟩^∞ M_i);
4  repeat
5    forall (M_i'[⟩^p M_i'') ∈ Waiting_i do
6      Waiting_i ← Waiting_i \ {(M_i'[⟩^p M_i'')};
7      synchpt ← false;
8      forall tf ∈ TF ∩ T_i, M_i''[tf⟩, ρ_i(M_i'', tf) ≥ ρ_i(M_i'', T_i \ TF) do
9        trysynch_i ← trysynch_i ∪ {(M_i'[⟩^p M_i'', tf)};
10       synchpt ← true;
11     endfall
12     forall t_i ∈ T_i \ TF, M_i''[t_i⟩M_i''', ρ_i(M_i'', t_i) = ρ_i(M_i'', T_i \ TF) do
13       if ¬synchpt ∨ρ_i(M_i'', t_i) = ρ_i(M_i'', T_i) then
14         NODE.ADD(M_i'[⟩^{min(p,ρ_i(t_i))} M_i''');
15         ARC.ADD(M_i''[t_i⟩^{ρ_i(t_i)} M_i''');
16       else
17         NODE.ADD(M_i''[⟩^{ρ_i(t_i)} M_i''');
18         ARC.ADD(M_i''[t_i⟩^{ρ_i(t_i)} M_i''');
19       endif
20     endfall
21   endfall
22 until stable ;
23 return trysynch_i
```

6 Conclusions

The modular state space technique [CP00] proves to give good results in practical cases [LP04,Pet05] to alleviate the state space explosion problem. This technique is designed to handle nets where modules communicate through transition fusion, i.e. synchronise. The technique is particularly efficient when the system modules exhibit strong cohesion and weak coupling.

In practice many systems use some kind of priority mechanism, either implicit or explicit. This is the case when there are timing constraints or when some events should be executed before others, once they are enabled (e.g. in scheduling problems). The priority used may either be static, i.e. it is fixed for a

Table 1. State space results for device message handling.

	Modular state space				Unfolded state space			
Devices	Nodes	Arcs	Sec	KB	Nodes	Arcs	Sec	KB
1	7	42	0.009	2.8	39	58	0.005	3.7
2	39	48	0.061	11.4	1441	3482	0.074	100.2
3	343	6,174	0.565	101.9	51,304	156,609	3.643	4,205.5
4	2,401	57,624	5.412	912.6	178,2011	6,264,028	196.140	164,439.5
5	16,807	504,210	51.856	7,902.5	—	—	> 3320	$> 2,570,457.13$

given transition and will never change during the life of the system, or dynamic, meaning that it does not depend solely on the transition involved, but also on the current marking.

In this paper, we have first introduced modular Petri nets with dynamic priorities and then adapted the modular state space technique to these nets. This involved defining the priority for synchronisation transitions in a way which is consistent with both practical and theoretical concerns. The resulting modular state space with priorities contains more states than necessary since part of the preemption due to priorities cannot be known *a priori*. Some preliminary results have been generated from a partial implementation. These results provide a proof of concept for the proposals. A fully-fledged implementation is required to produce more extensive comparative results.

One could consider other alternatives for the priority of synchronisation transitions. Motivated by a consideration of timed systems, we set the priority of synchronisation transitions to be the *minimum* of the priorities of the constituent transitions. Instead, it could be set to the *maximum*. In this case, the preemption rules would change — if the local component of a synchronisation transition had priority greater than that of a local transition, then preemption would always occur. On the other hand, if the local component of a synchronisation transition had lesser priority than that of a local transition, then preemption may still occur. This change would not affect the propositions, but would require changes to the logic of the presented algorithms.

Further work is required on both theoretical and practical issues. We should study the use of prioritised modular state spaces to prove system properties. In order to verify some properties, it will be necessary to locally unfold part of the state space, so as to get rid of the spurious states, while other properties will be directly verified on the modular structure. Then, we intend to apply the technique to a large case study. An appropriate example is the avionics mission system from [PKBQ03], which includes both timing constraints and explicit priorities of tasks to be scheduled on CPUs within a certain time frame. Another example with both timing constraints and priorities is the Fieldbus protocol [MSF+99], which provided the motivating example of message handling in Section 2.

References

[Bau97] Falko Bause. Analysis of Petri nets with a dynamic priority method. In Azéma, P. and Balbo, G., editors, *Proc. 18th International Conference on Application and Theory of Petri Nets, Toulouse, France, June 1997*, volume 1248 of *LNCS*, pages 215–234, Berlin, Germany, June 1997. Springer-Verlag.

[CP00] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *The Computer Journal*, 43(3):224–242, 2000.

[LP04] C. Lakos and L. Petrucci. Modular analysis of systems composed of semi-autonomous subsystems. In *Proc. 4th Int. Conf. on Application of Concurrency to System Design (ACSD'04), Hamilton, Canada, June 2004*, pages 185–194. IEEE Comp. Soc. Press, June 2004.

[LP07] C. Lakos and L. Petrucci. Modular state space exploration for timed Petri nets. *Journal of Software Tools for Technology Transfer*, 9(3–4):393–411, June 2007.

[Mäk02] M. Mäkelä. Model Checking Safety Properties in Modular High-Level Nets. In W. van der Aalst and E. Best, editors, *24th International Conference on the Application and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 201–220, Eindhoven, The Netherlands, 2002. Springer.

[MSF+99] A.B. Mnaouer, T. Sekiguchi, Y. Fujii, T. Ito, and H. Tanaka. Colored Petri nets based modeling and simulation of the static and dynamic allocation policies of the asynchronous bandwidth in the fieldbus protocol. In J. Billington and M. Diaz, editors, *Advances in Petri Nets*, volume 1605 of *LNCS*, pages 93–130. Springer-Verlag, 1999.

[Pet05] L. Petrucci. Cover picture story: Experiments with modular state spaces. *Petri Net Newsletter*, 68:Cover page and 5–10, April 2005.

[PKBQ03] L. Petrucci, L. M. Kristensen, J. Billington, and Z. H. Qureshi. Developing a formal specification for the mission system of a maritime surveillance aircraft. In *Proc. 3rd Int. Conf. on Application of Concurrency to System Design (ACSD'03), Guimarães, Portugal, June 2003*, pages 92–101. IEEE Comp. Soc. Press, 2003.