http://www-lipn.univ-paris13.fr/~petrucci/PAPERS
In Proc. International Workshop on Petri Nets and Software Engineering (PNSE'07, associated with Petri Nets'07), Sieldlce, Poland, University of Podlasie, pages 175-190, june 2007.

Modular State Spaces and Place Fusion ⋆

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
petrucci@lipn.univ-paris13.fr

Abstract. Modular state spaces [CP00,LP04] turned out to be an efficient analysis technique in many cases [Pet05]. However, it is designed for modules communicating through shared transitions. Since several models, e.g. Hierarchical Coloured Petri Nets [Jen94], rather use a place sharing mechanism, we investigated the possibility of adapting the modular state space technique to such models. This paper reports the different trials and experiments before drawing conclusions.

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. Experiments have indicated [Pet05] that modular analysis can produce a significant reduction in the size of the state space, particularly for systems where the modules exhibit strong cohesion and weak coupling. However, these techniques are designed for systems where modules share transitions, whereas some common formalisms (e.g. hierarchical CP-nets) use a place fusion mechanism. Our goal is then to benefit from modular techniques when place fusion is present.

After introducing the basic definitions in section 2, we recall, in section 3, the modular state space exploration technique from [CP00,LP04]. We then discuss, in section 4, different possibilities for adapting it to systems sharing places rather than transitions. These are evaluated, and lead to a solution to deal with such systems, explained and experimented in section 5.

⋆ This work is supported by the French-Australian Science and Technology programme 09872RF.

2 Basic Definitions

2.1 Petri Nets

We first recall the basic definitions and notations for Petri nets, their markings, enablings and occurrence rules:

Definition 1. *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* $\mathbb{N}$*, and* M_0 *is the* **initial marking**, *namely a function mapping from* P *into* $\mathbb{N}$.

Definition 2. *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 *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 *is enabled in a marking* M_1*, it may* **occur**, *changing the marking* M_1 *to another marking* M_2*, defined by:* $\forall p \in P : M_2(p) = (M_1(p) - W(p,t)) + W(t,p)$*. The set of markings* **reachable** *from a marking* M *is:* $[M\rangle = \{M' \mid \exists \sigma \in T^* : M[\sigma\rangle M'\}$.

2.2 Modular Petri Nets

Modular Petri nets are defined in a similar manner. As in the definitions of [CP00] we consider communication through places as well as transitions.

Definition 3. *A* **modular Petri net** *is a triple* $MN = (S, PF, TF)$*, satisfying:*

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.*
2. $PF \subseteq 2^P \setminus \{\emptyset\}$ *is a finite set of non-empty* **place fusion sets** *such that:*
   - *For nodes* $x \in P \cup T$ *we use* $S(x)$ *to denote the module to which* x *belongs. For all* p *in* P *we define* $M_0(p) = M_{0_{S(p)}}(p)$.
   - *Members of a place fusion set have identical initial markings:*

$$\forall pf \in PF : \forall p_1, p_2 \in pf : [M_0(p_1) = M_0(p_2)].$$

3. $TF \subseteq 2^T \setminus \{\emptyset\}$ *is a finite set of non-empty* **transition fusion sets**.

In the following, TF also denotes $\cup_{tf \in TF} tf$.
We now introduce place groups and transition groups.

Definition 4. *A* **place group** $pg \subseteq P$ *is an equivalence class of the smallest equivalence relation containing all pairs* $(p_1, p_2) \in P \times P$ *where:*

$$\exists pf \in PF : p_1, p_2 \in pf.$$

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 place groups is denoted by PG*, and the set of transition groups by* TG*.*

A place is a member of exactly one place group, which represents e.g. a shared resource. Place groups form a partition of the set of places. 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 place groups and transition groups, i.e. $\forall pg \in PG, \forall tg \in TG$:

$$W(pg, tg) = \sum_{\substack{p \in pg \\ t \in tg}} W(p, t), \; W(tg, pg) = \sum_{\substack{p \in pg \\ t \in tg}} W(t, p).$$

Markings of modular Petri nets are defined as markings of Petri nets, over the set PG of all place groups. 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 5. *A transition group* tg *is* **enabled** *in a marking* M*, denoted by* $M[tg\rangle$*, iff:*

$$\forall pg \in PG : W(pg, tg) \leq M(pg).$$

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:*

$$\forall pg \in PG : M_2(pg) = (M_1(pg) - W(pg, tg)) + W(tg, pg).$$

Example: Figure 1 depicts a modular PT-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. Note that in this example, there is no place fusion set.

2.3 State Spaces of 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.

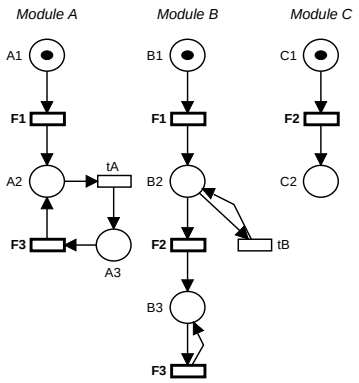

Fig. 1. Modular PT-net with modules A, B and C.

Definition 6. *Let $PN = (P, T, W, M_0)$, be a Petri net. The* **State Space** *of PN is the directed graph $SS = (V, A)$, where:*

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

Example: The (full) state space for the modular PT-net of figure 1 is shown in figure 2. 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.

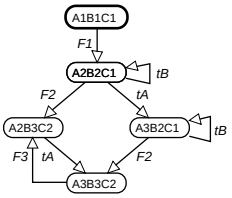

Fig. 2. The full state space of the system in figure 1.

When considering the modular state space, as well as checking properties of the system, we will use Strongly Connected Components. The set of all strongly connected components is denoted by SCC. For a node v and a component $c \in SCC$ we use $v \in c$ to denote that v is one of the nodes in c. A similar notation is used for arcs. We use v^c to denote the component to which v belongs.

3 Modular State Spaces and Transition Fusion

In this section, we consider modular Petri nets with transition fusion only, i.e. $PF = \emptyset$.

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.

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.

For any reachable marking M, we use $M^{\mathscr{q}}$ to denote the product (or tuple) of Strongly Connected Components (SCCs) M_s^c of the individual modules:

$$\forall M \in [M_0\rangle : M^{\mathscr{q}} = \prod_{s \in S} M_s^c.$$

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

Definition 7. *Let $MN = (S, TF)$ be a modular Petri net with the initial marking M_0. The* **modular state space** *of MN is a pair $MSS = ((SS_s)_{s \in S}, SG)$, where:*

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

Explanation:
(1) The definition of the state space graphs of the modules is a generalization 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 enabled internal transitions of the module.
(2) Each node of the synchronisation graph is labelled by a $M^{\mathscr{q}}$ and is a representative for all the nodes reachable from M by occurrences of local transitions only, i.e. $[[M\rangle$. 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.

Example: The modular state space for the modular PT-net of figure 1 is shown in figure 3. Note that there is a local state space for each module, as well as a synchronisation graph which captures the occurrence of fused transitions. We do not distinguish between nodes and SCCs since, in this case, all SCCs consist of a single node (which is seldom the case in practice).

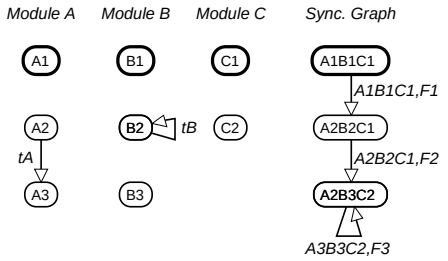

Fig. 3. The modular state space of the system in figure 1.

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.

4 Modular State Spaces and Place Fusion

Some tools, e.g. CPNTools [CPNa], allow for place fusion instead of transition fusion. Therefore, one of our goals is to evaluate the possibility of adapting the modular state space technique to cater for modular Petri nets with place fusion only. Hence, in this section, we consider modular nets such that $TF = \emptyset$.

4.1 Problems due to Place Fusion

The composition of state space graphs is more complex when sharing places rather than transitions. This can be seen on the example of figure 4, where the grey place p_2, initially empty, is the shared one: in this case, we are guaranteed that if at least one of the modules has an infinite state space graph, the Modular

PT-net also has an infinite state space graph. But it is impossible to tell anything about the state space graph of the modular PT-net if those of the modules are finite. This is due to the fact that a module can provide enough tokens in a place fusion set to allow some transitions, in another module, to be enabled. And then this second module can provide some more tokens for the first one and so on.

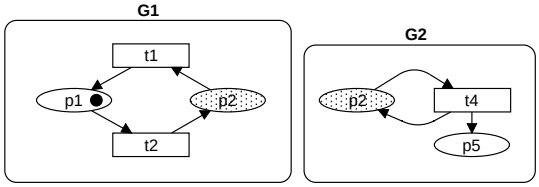

Fig. 4. Two modules each with finite graphs, but a Modular PT-net with infinite graph.

Several approaches to state spaces for modular nets with place fusion have been considered. In the following subsections, we briefly describe their pros and cons.

4.2 Dedicated Algorithm to construct the State Space

The first approach consists in designing a dedicated algorithm to construct the state space. The background considered was both the technique with transitions fusion explained in section 3 [CP00,LP04] and the compositional construction of the covering graphs from [FP92,FP94].

To meet efficiency, memory usage and properties verification methods similar to those provided by Modular State Spaces, the following requirements should be satisfied:

- The state space must not be the complete flat state space but a collection of state spaces or abstractions of these.
- All behaviour and reachable states should be represented.
- An unfolding to the flat state space must be possible, which could serve as a basis for property verification.

When building a modular state space, the fused places can either be part of local state space graphs, or of a global structure similar to the synchronisation graph for transition fusion.

Let us consider the case where the shared places are considered in the local graphs. The graphs can be build step by step, as in [FP92], but it is necessary to keep track of the synchronisation points between the modules. An identical marking is not sufficient, as shown in the example of figure 6, corresponding to modules A and B in figure 5 where the dotted arrows indicate that a move in the other module changes the local state. In this example, finding state A_2B_2 from the local modules is not obvious.

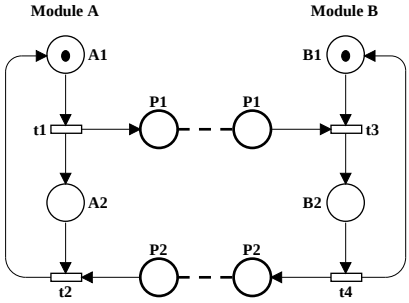

Fig. 5. Example with shared places

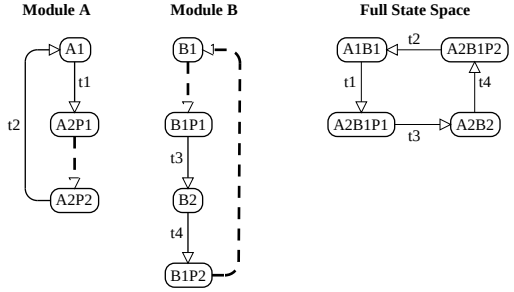

Fig. 6. Shared places in local state spaces

Hence, we consider the construction of a global structure to handle the shared places and synchronisations between modules. A small example is given in figure 7. This structure contains the shared places and the transitions they are connected to, plus the corresponding markings in the local state spaces. One can note that the structure obtained is similar to what we would get if the modular net was using transition fusion only with one module containing all fused places and their connected transitions.

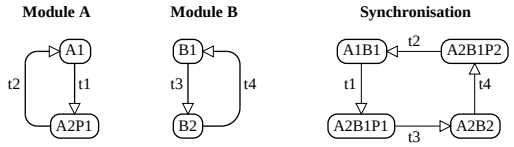

Fig. 7. Shared places in global structure

4.3 Transforming Place Fusion into Transition Fusion

Therefore, designing a specific algorithm to handle place fusion does not appear worthwhile. Instead, our approach leads us to consider an appropriate transformation from a modular net with place fusion to a modular net with transition fusion only, in order to apply the modular state space technique.

Transforming place fusion into transition fusion can be achieved in many ways. However, as shown in [Pet05], the benefits of the modular state space technique depend on the level of coupling between modules. In the following section, after indicating different approaches, we show some experimental results and then derive guidelines as to how to proceed.

5 From Place Fusion to Transition Fusion

We first present different possibilities to transform a general modular net (i.e. with place fusion and transition fusion) into a modular net with place fusion only. The fused transitions remain fused, so we will focus on the fused places.

5.1 Different possibilities

Isolating fused places A first and rather straighforward way of dealing with fused places is to isolate them into a single or several dedicated modules. In the first case, all fused places are grouped into a single module together with their connected transitions. These transitions become shared transitions, that have to be fused with a corresponding copy in their origninal module. The second case is similar, but places can be scattered in several modules which contain only fused places and their connected transitions.

Example: The example of figure 5 can be transformed into a modular net with one module for shared places (figure 8) or one module per shared place (figure 9).

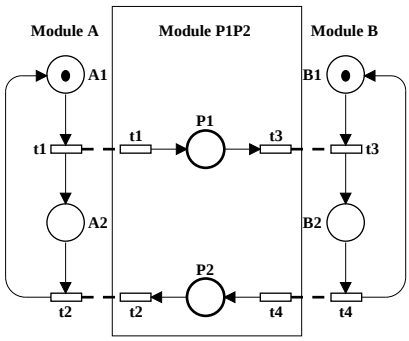

Fig. 8. Shared places in a single module (net `1module`)

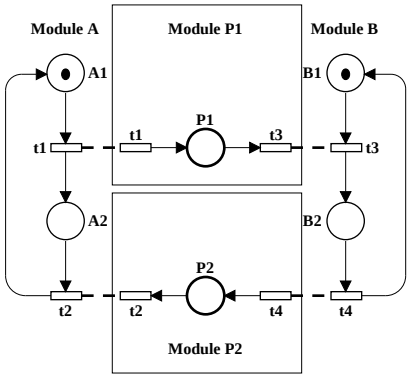

Fig. 9. Shared places in separate modules (net `2modules`)

However, as we shall see in section 5.2, this approach, although simple, is not very efficient. Indeed, all the activity of the module(s) holding only fused places only happens in the synchronisation graph, i.e, as shared activity.

Integrating fused places within modules In order to avoid modules without any local activity, it is desirable to keep the fused places in one of their original modules. The transitions connected to the place which are not already in the module lead to a new copy which is fused. The place is removed from the other modules, and the transitions they are connected to become shared. As the original place has a copy in several modules, the question is: in which module should the place remain. We will see, in section 5.2, that the different possibilities can be more or less efficient.

Example: For the example of figure 5, both places P1 and P2 can be integrated within module A (figure 10) or module B. Otherwise one place can be integrated in each module, i.e. P1 in module A and P2 in module B (figure 11) or vice versa.

5.2 Experiments

Before presenting results for an elaborate case study, we first show the experimental results for the small examples from section 5.1. Even though the modular state spaces are very small and present slight differences (see table 1), they are quite relevant to what happens more generally. They already show that integrating a place to one of its original modules might give better results.

We have also experimented with a more extensive case study — that of a sliding window protocol. A version of the protocol is depicted in figure 12. This is adapted from the stop-and-wait protocol in [Jen94], where places A, B, C and D connected the various modules by place fusion. We have extended that

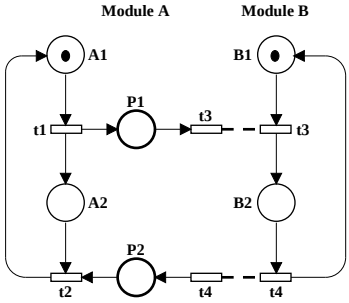

Fig. 10. Both places integrated in module A (net `AP1P2`)

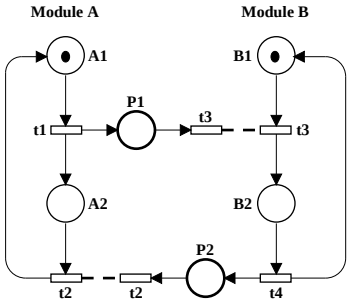

Fig. 11. One place integrated in each module (net `AP1BP2`)

protocol to a sliding window protocol, and have used transition fusion so as to be amenable to modular state space exploration.

This version of the protocol was analysed as a timed system using the *Maria* reachability analyser [MÖ2]. *Maria* supports modular analysis but not time. Consequently, the analysis of this timed system was achieved by incorporating a generic timing infrastructure. A description of that timing infrastructure together with the associated results are reported elsewhere [LP07]. Here, we consider different modular decompositions of the system in line with the proposals considered earlier in this paper.

Briefly, the protocol operates as follows. It is composed of four modules representing the *sender*, the *message channel* of the network, the *acknowledgement*

Example	`1module`	`2modules`	`AP1P2`	`BP1P2`	`AP1BP2`	`AP2BP1`
Nodes	11	12	9	8	8	9
Arcs	4	4	5	4	4	5

Table 1. Modular State Space sizes for the small example

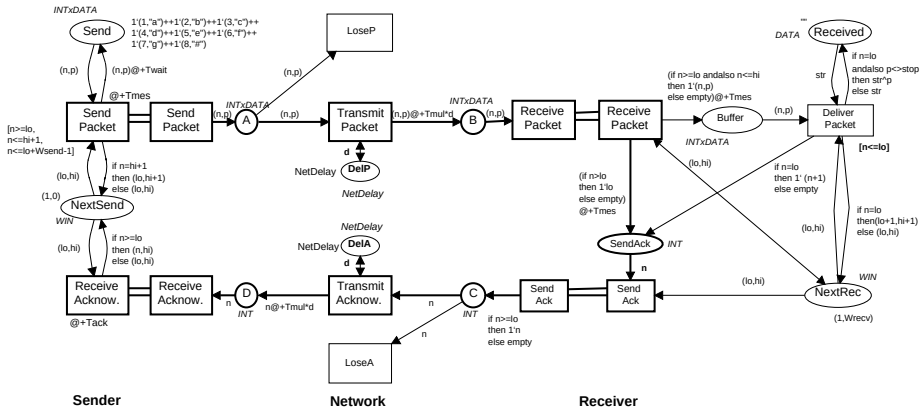

Fig. 12. Timed sliding window protocol.

channel of the network and the *receiver*. These are displayed on the left-hand side, the center and the right-hand side of the figure, respectively. Here, the four modules communicate with their neighbours through fused transitions: `Send Packet`, `Receive Packet`, `Send Ack` and `Receive Ack`.

The *sender* module can only send packets or receive acknowledgements. The packets to send are initially stored in place `Send`. They are represented by a pair `(n,p)` where n is the packet number and p the data. We can refer to a packet consisting of the sequence number 1 and data *"a"*, available at time 0 by the notation *(1,"a")@0*. The sending operation takes some time as indicated by the `@+Tmes` label attached to transition `Send Packet`. When a packet is sent, its token remains in place `Send`, but its timestamp is incremented by the time-out retransmission delay `Twait`. With `Twait` set to 80 and `Tmes` set to 8, the token *(1,"a")@0* would be replaced by *(1,"a")@*`Tmes+Twait`*=(1,"a")@88*. The packets that can be sent must have a number falling between the lowest not yet acknowledged (variable `lo`) and the first non-sent packet (`hi`+1). Moreover, the difference between these two bounds cannot exceed the window size `Wsend`. These conditions are all gathered in the guard of transition `Send Packet`. The lower and upper bounds are stored as a pair `(lo,hi)` in place `NextSend`.

The reception of an acknowledgement (transition `Receive Ack`) takes some time specified by a constant `Tack`. This is expressed by attaching `@+Tack` to the transition. The values of the bounds in place `NextSend` are updated if a message with a number `n` greater than `lo` is acknowledged. (Note that in this simple case study, we do not consider cyclic sequence numbers.)

The *network* stores packets sent in place `A`. Then, it can either lose them (transition `LoseP`) or transmit them (transition `Transmit Packet`). In that case, a delay corresponding to the time spent for transmission is applied to the packet, denoted $(n,p)@+Tmul*d$. The packet is then ready to be received. A similar scheme is applied to the acknowledgements. The main difference is that an acknowledgement is put in the network only if the packet `n` has not yet been ac-

knowledged. This is indicated by the term associated with the arc from transition `Send Ack` to place `C`.

Finally, the module *Receiver* can receive packets numbered from the first expected one (`lo`) up to the size of the reception window (`hi`=`lo`+`Wrecv`-1). (Note that this allows for packets to arrive out of sequence.) Such a received packet is stored in place `Buffer` until the processing time `Tmes` has elapsed. Moreover, if the message is not the first one expected (`n>lo`), an acknowledgement numbered `lo` is prepared in place `SendAck`. This informs the sender that the number of the next message expected is `lo`. When a packet is in the receiver buffer, it can effectively be accepted and processed, via transition `Deliver Packet`. If it has a sequence number less than `lo`, then it has already been delivered and this is a duplicate that needs to be discarded. If its sequence number is equal to `lo`, then it is delivered and the receiving window is advanced — the contents `p` of the packet are concatenated with the contents previously received, thus forming a string stored in place `Received`; the receiving window is updated by incrementing both bounds; and an acknowledgement for the new lower bound `(n+1)` is prepared. When an acknowledgement is sitting in place `SendAck`, it can be transmitted to the *sender* via the *network* module.

In order to give intuitive names to the various modular decompositions, we use the following abbreviations:

Snd The Sender, including transitions *Send Packet*, *Receive Acknow* and the neighbouring places, but not including places *A* and *D*.
Rcv The Receiver, including transitions *Receive Packet*, *Deliver Packet*, *Send Ack* and the neighbouring places, but not including places *B* and *C*.
Msg The Message Channel, including transitions *LoseP*, *Transmit Packet*, and the neighbouring places, but not including places *A* and *B*.
Ack The Acknowledgement Channel, including transitions *LoseA*, *Transmit Acknow*, and the neighbouring places, but not including places *C* and *D*.

We have then experimented with the following configurations:

Prot 3: One reference configuration with modules {Snd}, {A, Msg, B}, {Rcv}, {C, Ack, D}.
Prot 3b: Another reference configuration with modules {Snd, A, Msg, B}, {Rcv, C, Ack, D}.
Prot 3d: Variant of *Prot 3* with places B and D in modules by themselves, i.e. modules are {Snd}, {A, Msg}, {B}, {Rcv}, {C, Ack}, {D}.
Prot 3e: Variant of *Prot 3b* with places B and D in modules by themselves, i.e. modules are {Snd, A, Msg}, {B}, {Rcv, C, Ack}, {D}.
Prot 3g: Variant of *Prot 3d* with both places B and D in the same module, i.e. modules are {Snd}, {A, Msg}, {B, D}, {Rcv}, {C, Ack}.
Prot 3h: Variant of *Prot 3e* with both places B and D in the same module, i.e. modules are {Snd, A, Msg}, {B, D}, {Rcv, C, Ack}.

We have run two sequences of tests. The results in figure 13 are for a sliding window size of 1, while the results in figure 14 are for a sliding window size of 2. In both cases, the total simulation time ranges from 60 to 200 time units.

We make the following observations:

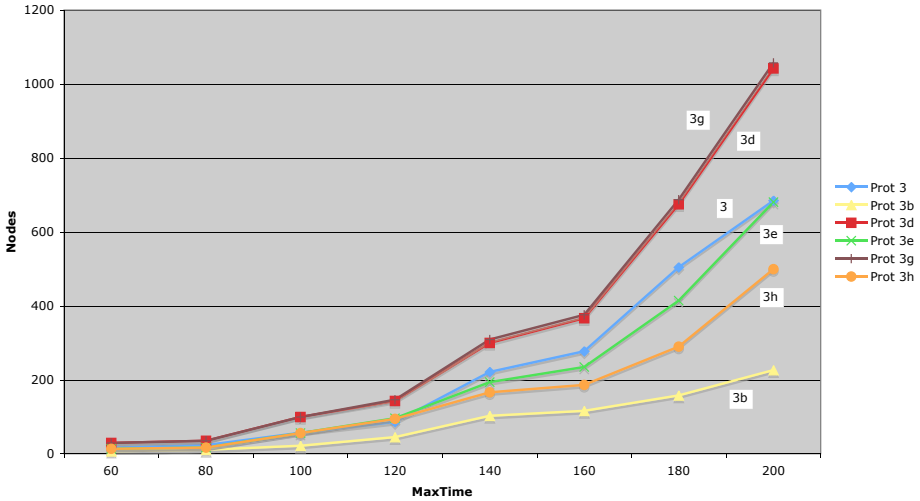

Fig. 13. Variants with sliding window size of 1.

- As in the earlier published results [LP07], version 3b has a much smaller state space than that of version 3. It takes advantage of the cohesion between the Sender and the Message Channel, and the relatively loose coupling with the other components. It is clear that this is still optimum amongst the other variations we have since considered.
- Version 3d is significantly worse than version 3, and similarly for versions 3e and 3b. In other words, putting the places B and D in modules by themselves results in a significantly larger synchronisation graph, to the extent that only smaller problem sizes can be explored. This is not too surprising because both addition and removal of tokens to these places will now add nodes and edges to the synchronisation graph.
- Versions 3d and 3g are almost identical, i.e. places B and D are essentially independent. Whether they are placed in separate modules or in the one module makes little difference to the size of the synchronisation graph.
- Versions 3e and 3h exhibit some difference, i.e. places B and D are no longer independent because of the merging of the channels with the Sender and Receiver.

Further experiments have been performed, giving similar results. They are not presented here. From all these experiments, in the next section, we derive guidelines for transforming place fusion into transition fusion.

5.3 Guidelines

As shown in the results of table 1 and figures 13 and 14, creating a specific module for shared places is the worst solution: the number of nodes in the modular state

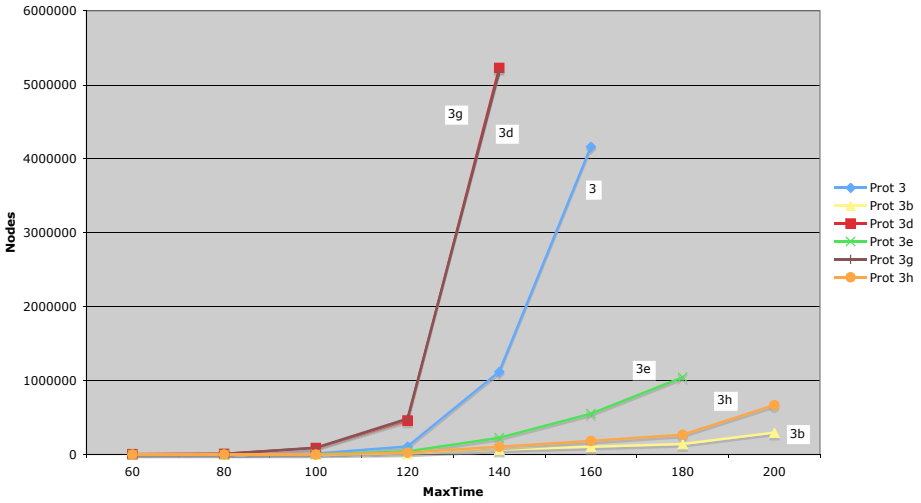

Fig. 14. Variants with sliding window size of 2.

space is large. This is due to the extreme situation of the new module(s) having no internal behaviour. Hence, all the behaviour concerned with these places is shown in the synchronisation graph.

On the contrary, when the shared places are incorporated within one of their original modules, the size of the modular state space is much smaller. Here, some of the related behaviour is then local, and less synchronisation is represented in the synchronisation graph.

The choice of optimum module will depend on the connectivity with the place — the greater the connectivity, the better the results when including the place in that module. However, our experiments show that best results are obtained when the question of where to instantiate the fused place is superceded by the question of how to maximise cohesion and minimise coupling. In the protocol example, our best results were obtained by combining the sender module and the message channel. Thus, sending a message and then losing it becomes purely local activity.

6 Conclusions

A lot of Petri nets, designed with modular concepts, are structured using a mechanism similar to place fusion. This is in particular the case for Hierarchical Coloured Petri nets, which are quite popular [CPNb] due to their tool support (Design/CPN [CPNc] and CPNTools [CPNa]).

The modular state space technique [CP00] proves to give good results in practical cases [LP04,Pet05] to alleviate the state space explosion problem. However, this technique is designed to handle nets where modules communicate through transition fusion.

In this paper, we have investigated the possibility of using modular state spaces when modules communicate through place fusion. We first investigated the possibility of designing an ad-hoc algorithm. It turned out that this would lead to a complex design and to results very close to those obtained with transition fusion. Therefore, the work was refocussed on transforming a modular net with place fusion into a net with transition fusion only. Several tranformation schemes were studied, using examples ranging from simple to rather elaborate. The sizes of the different modular state spaces were analysed to understand which characteristics are involved in getting better results.

This study led to some guidelines which could be partially automated. However, they are quite general and should not be considered as always giving the best results. Putting fused places into modules by themselves or with other fused places gives the worst results. The best results are obtained not by simply focussing on where to put the fused places, but by carefully considering how to maximise cohesion and minimise coupling between the modules.

References

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

[CPNa] *cpntools*. `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`.

[CPNb] *Examples of Industrial Use of CP-nets*. `http://www.daimi.au.dk/CPnets/intro/example_indu.html`.

[CPNc] Design/CPN *online*. `http://www.daimi.au.dk/designCPN`.

[FP92] A. Finkel and L. Petrucci. Avoiding state explosion by composition of minimal covering graphs. In *Proc. 3rd Int. Workshop Computer Aided Verification (CAV'91), Aalborg, Denmark, July 1991*, volume 575 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 1992.

[FP94] A. Finkel and L. Petrucci. Propriétés de la composition/décomposition de réseaux de Petri et de leurs graphes de couverture. *RAIRO Informatique Théorique et Applications*, 28(2):73–124, 1994.

[Jen94] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and practical use. Volume 2: analysis methods*. Monographs in Theoretical Computer Science. Springer, 1994.

[LP04] C. Lakos and L. Petrucci. Modular analysis of systems composed of semiautonomous 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*, 2007. To appear.

[MÖ2] 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.

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