New Petri Net Programming Features in PEP*

Cécile Bui Thanh[1] and Christian Stehno[2]

[1] Université Paris 12, LACL, 61 avenue du général de Gaulle F-94010 Créteil, France.
bui@univ-paris12.fr
[2] Department of Computing Science, Carl von Ossietzky Universität, D-26111 Oldenburg, Germany.
stehno@informatik.uni-oldenburg.de

Abstract. We present two new facilities of the high level Petri net editor of the PEP tool. The latest version extends the class of supported Petri nets by a time extension of M-nets. Additionally it features a new operator for asynchronous communication complementing the synchronization operator. We present an example of an ARQ protocol with enhanced acknowledgment handling. form.

1 Introduction

The PEP tool [5, 9] provides a development environment for a number of different parallel programming and specification languages. Additionally, the high and low level Petri net levels offer another, quite unique, programming interface. Especially the high level nets facilitate design of data handling algorithms and complex programs through their extended token types and powerful set of operators.

The latest extensions of the net editors add major new features with the introduction of a new basic operator tie and a new class of nets, namely the Time Petri net extension of the already used M-nets.

This paper presents the enhancements of the specification and programming process gained by these extensions. The example chosen for this purpose is a simple ARQ (automatic repeat request) protocol with possibly deferred acknowledgements.

The next two sections will introduce the formalisms used for the example, i.e. (Time) M-nets and the tie operator. In Sect. 4 the example is presented, and finally in Sect. 5 the paper is concluded and some future work is highlighted.

2 M-nets and Time M-nets

We will only give a short introduction to M-nets and their time extension. More details can be found in the cited papers and related articles.

* This work has been done during a three month visit of one of the authors to Oldenburg, and has been partially supported by the Procope project PORTA (Partial Order Real Time Semantics)

The coloured Petri nets algebra of M-nets (multi labelled nets [2]) has been developed as a flexible semantic model for concurrent programming languages. The annotations of the net elements and a set of operations provide a framework for the compositional creation of complex nets. Operations include sequential, parallel and alternative composition, iteration and synchronous communication by synchronisation and restriction. Unfolding into plain P/T nets provides possibilities for a formal verification of the designed systems by means of well-known algorithms for ordinary Petri nets, e.g. with the PEP tool.

Time M-nets (TM-nets) are a cautious, but still substantial extension of M-nets towards real-time systems. The time extension was originally defined in [4] for a real-time version of the specification language SDL. TM-nets had been supported so far only in the MOBY tool [1] without a possibility to validate the nets. The newly implemented TM-net features of the PEP tool, together with the already presented analysis tools for timed systems [12, 3] increase the importance of TM-nets by far.

The time extension is done analogously to the Time PBC extension in [7]. Transitions get an additional inscription of an interval in the natural numbers. This interval restricts the ability of the transition to fire according to [8], i.e. the lower (upper) bound defines the earliest (latest) firing time of each transition.

To preserve the properties of M-nets with respect to unfolding into low level nets for TM-nets, firing times are not unique to a transition, but each firing mode (i.e. binding enabling the transition) counts time steps on its own, after the mode is enabled. As such, each transition may have a large number of clocks showing different times.

The current implementation in PEP covers just an elementary version of TM-nets. We restrict the interval bounds to natural numbers (plus infinity for the upper bound). This is in contrast to the original definition where arbitrary expressions are allowed, which still have to evaluate to natural numbers according to the chosen binding, though. As such, the restriction is not just syntactically, but the expressiveness of the simple class is still large enough for many real world systems.

3 Tie operator

The tie operator was introduced to M-net algebra to allow asynchronous communications [6] in a compositional way. This extension to M-Net algebra allowed to give a complete semantics of $B(PN)^2$[3] [?] in terms of M-net, and a simpler semantics for FIFO buffers [10, 11].

To use asynchronous communications with M-nets, they are extended by new transition labels, called *link labels*. These labels indicate that a transition can export and/or import information from an asynchronous channel.

We consider a set B of *tie symbols* such that each $b \in \mathsf{B}$ has a type $\mathsf{type}(b)$. An *asynchronous link* $b^d(\tilde{a})$, where $b \in \mathsf{B}$, $d \in \{+, -\}$ and $\tilde{a} \in \mathsf{type}(b)$, represents

[3] Basic Petri Net Programming Notation

an asynchronous communication action on (the channel represented by) b. The direction of the communication (export or import) is given by d, and $\tilde{a}$ is a set of arguments representing possible information communicated on b. A link label is a multi-set of asynchronous links.

An example drawn with the high-level net editor of the PEP tool is given in Fig. 1. On the left hand, we show a net N with asynchronous annotations. The label {`b+(x)`} on transition T1 means that this transition *can* export $x \in \mathsf{type}(b)$ on a channel b, and {`b-(1),b+(1)`} on T2 means that T2 expects to import a token labelled "1" and to export it again via b. The result of N tie b is given on the right hand of the figure: The asynchronous links of N containing b were made effective by the operation, i.e. the transitions containing these asynchronous links now export and/or import the described informations when they are fired, according to the directions given by the previous notation. Every exchanged information goes through the created buffer place of b (named P6) with $\mathsf{type}(b)$ as type. Note that applying tie b deletes the link labels concerning b (not the other links) from every transition.

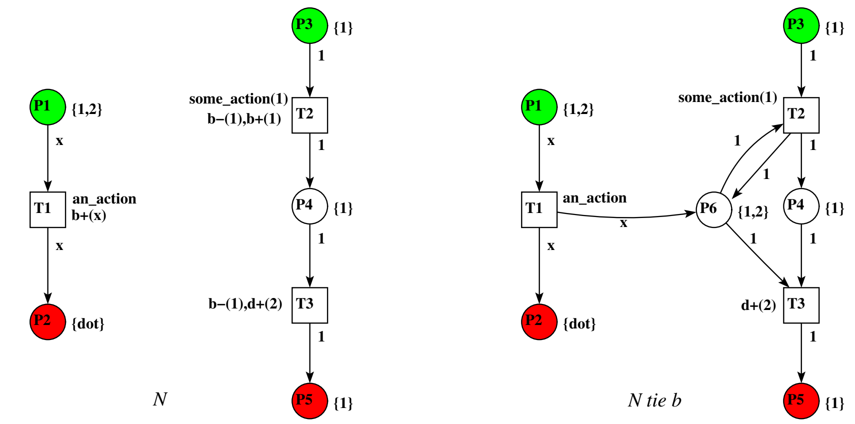

Fig. 1. An example of using tie with the high-level editor. On the left hand: a High-level net N containing link labels, on the right hand: the same net after applying tie b

In the high-level editor, declarations (respectively, deletion, modification) of asynchronous links names and their corresponding types are made through the window of the tie operator (cf. Fig. 2). The syntax for link types is the same as for place types. Link labels on transitions are added the same way as action multisets. After applying the operator once to some link b, this one is taken account as a brand new link and may be reused afterwards to add new annotations on link labels and to apply tie b again. This would then create a new buffer place for b each time the operation is called.

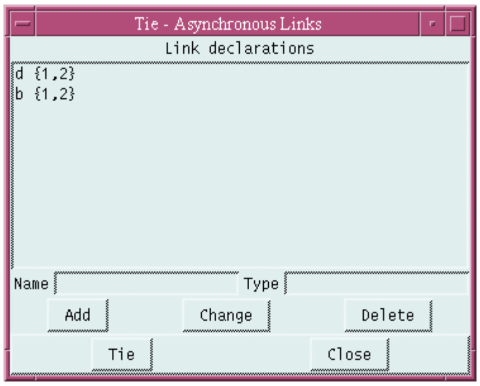

Fig. 2. The window of the tie operator

The tie operator was also defined for PBC (see [6] for details). This is not completely implemented yet, but after applying tie to each link, a net does not contain asynchronous notations anymore, so there is a way to unfold and analyse it right away.

4 Example of an ARQ protocol

We chose as an example for the presentation of the newly introduced features a modelisation of an ARQ protocol. There is a sender and a receiver side, where the sender communicates data packets to the receiver, who may answer with positive or negative acknowledgements. Both parts can be found in Fig. 3, with the sender at top and the receiver at bottom.

Communication is done in both directions via a buffered channel with some delay, i.e. asynchronous. Moreover the acknowledgements may be deferred, such that not every packet will be acknowledged, but a positive acknowledgement includes all waiting ack's in between. There is no buffering of correctly received data, so negative acknowledgements also clear everything backwards up to the negatively acknowledged packet.

If the sender receives a negative acknowledgement, it sets the next packet pointer to the received number, such that the next packet send will be the requested one. Positively acknowledged packets just increase the counter, with some sanity check done at reception. Sending data just incorporates the packet number, i.e. data content is abstracted away.

At the receiver side, things are even easier and more simplified. The received packets are stored in the respective counter place, which serves as a source for generating (positive or negative) acknowledgements. To model the removal of destroyed packets by some negative acknowledge, the received packet counter is reset accordingly. Thus, these packets will not be used either by the send ack transition, preventing false acknowledgements.

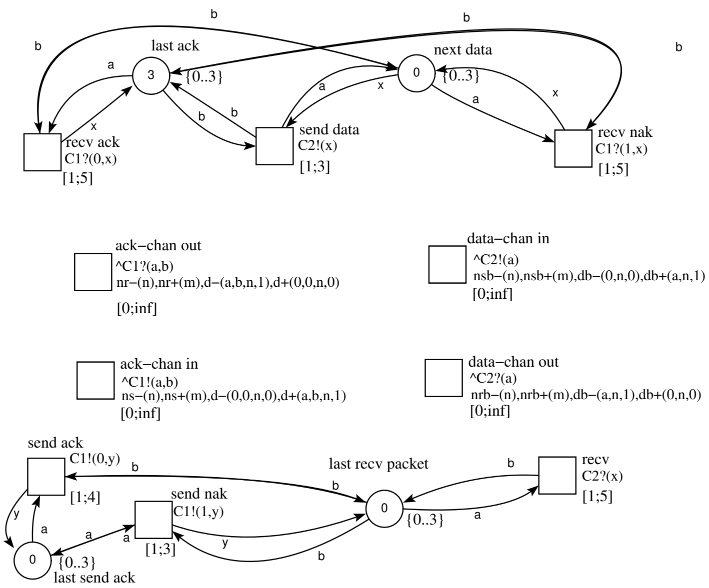

Fig. 3. ARQ model before synchronisation and tie

The two channels (C1 from receiver to sender and C2 vice versa) are simplified versions of the FIFO buffers presented in [10, 11]. Unused parts have been deleted, the data handling is simplified and the channel use has been extended with timing restrictions. Figure 3 just shows four transitions, which will create two separate ring buffers with FIFO semantics and compatible interface after application of the tie operation. Two additionally needed initialisation transitions are not shown, as they are just needed for technical reasons.

The time intervals are put to each channel operation, specifying some minimal and maximal delay needed for these actions. The channel transitions itself do not contribute to the delay with an interval of $[0, \infty]$. The delays have been chosen arbitrarily for this example, resulting in a delay for sending a packet between two and 8 time units, a complete round-trip with packet sending and acknowledgement takes up to 17 time units in the case of no data loss in between. The worst case scenario will take infinitely many time units, as we do not take into account fairness.

5 Conclusion and future works

We have presented the latest features of the PEP tool. This new version enables verification of Time M-nets and adds facilities for modelisation of asynchronous links in a compositional way. Moreover, these enhancements contribute to readability and handiness of the editor. M-nets provide a very powerful specification formalism. The ARQ system includes some interesting features, using just a small number of net elements. The time extension presented offers a comfortable high-level programming interface for real-time systems.

However, the asynchronous links have not yet been integrated into every part of the tool, which is planned for one of the next versions. This will remove some restrictions currently imposed on the user.

Time Petri nets are still not supported the same way like ordinary Petri nets. Most notably, a timed simulator is missing. Further work will include different time semantics and time support in even higher specification concepts, like SDL. Some further investigation of case studies of real-time systems, including formal verification is planned.

References

1. Peter Amthor, Hans Fleischhack, and Josef Tapken. MOBY - More than a tool for the verification of SDL-specifications. Technical report, Fachbereich Informatik, Carl von Ossietzky Universität Oldenburg, 1996.
2. E. Best, W. Frączak, R. P. Hopkins, H. Klaudel and E. Pelz. *M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages.* Acta Informatica, 35. Springer, 1998.
3. Hans Fleischhack and Christian Stehno. Computing a Finite Prefix of a Time Petri Net. In *ICATPN*, 2002. To appear.
4. Hans Fleischhack and Josef Tapken. An M-net semantics for a real-time extension of μSDL. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313, pages 162–181. Springer-Verlag, 1997.
5. B. Grahlmann. *The PEP Tool.* Computer Aided Verification, LNCS 1254. Springer, 1997.
6. H. Klaudel and F. Pommereau. *Asynchronous links in the PBC and M-nets.* ASIAN'99, LNCS 1742. Springer, 1999.
7. Maciej Koutny. A Compositional Model of Time Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *LNCS*, pages 303–322. Springer-Verlag, 2000.
8. P. Merlin and D. Farber. Recoverability of Communication Protocols – Implication of a Theoretical Study. *IEEE Transactions on Software Communications*, 24:1036–1043, 1976.
9. The PEP tool. http://parsys.informatik.uni-oldenburg.de/~pep
10. F. Pommereau. *FIFO buffers in tie sauce.* DAPSYS'00. Kluwer Academic Publishers, 2000.
11. Franck Pommereau and Christian Stehno. Fifo buffers in hot tie sauce. Technical Report 2001-04, LACL, Université Paris 12, 61 avenue du général de Gaulle, 94010 Créteil, France, 2001.
12. Christian Stehno. Real-Time Systems Design with PEP. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 476–480. Springer-Verlag, 2002.