Verifying Telecommunication Protocols with PEP

Bernd Grahlmann[a]

Institut für Informatik
Universität Hildesheim
Marienburger Platz 22
D-31141 Hildesheim
Germany
bernd@informatik.uni-hildesheim.de

Abstract

Nowadays telecommunication services become more and more complex. Because of the increasing comfort of these services the design and especially the verification of the underlying protocols is an extremely difficult task. In general protocols are only extensively tested but not verified due to the lack of powerful tools.

Within the **PEP** project a tool[b] has been developed which is well suited to design telecommunication protocols with a parallel specification and programming language called $B(PN)^2$. Furthermore the automatic translation into Petri nets allows the efficient verification of various interesting properties by use of a partial order based model checking algorithm.

Keywords: Protocols, Petri nets, Verification, Model checking, Tool.

1 Introduction

At the present time the design of parallel systems is a widespread task. In most areas it is necessary to ensure the correctness of the modelling. For example telecommunication networks are required to be highly reliable. Thus there is a big need for formal verification. Because of the complexity of telecommunication services the underlying protocols can only be tested extensively but not necessarily in a sufficient way to guarantee full correctness.

The **PEP** system offers a possible solution to this problem. Within the European project CALIBAN[c] a parallel specification and programming language called $B(PN)^2$[d] [3] has been developed to model and verify distributed systems. The **PEP** project has developed a tool which is well suited to design telecommunication protocols with $B(PN)^2$.

The integration of automatic translators for $B(PN)^2$ which generate the Petri net semantics allows the verification of various interesting properties given as branching time or linear time temporal logic formulae in terms of the program or the corresponding net.

---

[a] The **PEP** (Programming Environment based on Petri nets) project is financed by the DFG (Deutsche Forschungsgemeinschaft).

[b] A ftp version is available. Please send me an email

[c] CALculI BAsed on Nets

[d] Basic Petri Net Programming Notation

This is done by a model checker [6] integrated in the **PEP** tool. Its efficiency is based on the use of partial order semantics instead of constructing the state graph.

Within the CoCoN-project[e] a typical call handling for intelligent networks [7] which is closely related to a Basic Call State Model [5, 4] of the ITU-T (former CCITT) standardization has been designed and proven to be deadlockfree [9].

This paper shows that **PEP** can be used for the design of similar protocols. It is structured as follows. In section 2 a brief description of the different kinds of objects which are used by the **PEP** tool for the modelling and the verification is given. In section 3 the idea of the telecommunication protocol is explained. The next section describes how a part of the network can be modelled in $B(PN)^2$. In section 5 a translation into Petri nets is given. The verification possibilities of the **PEP** system are explained in section 6. In the last section a conclusion and an overview over future work are given.

2 Objects used by the PEP system

The **PEP** system uses different kinds of objects for the modelling and the verification of parallel systems. The modelling of parallel systems is typically done by writing programs in the predicative programming language $B(PN)^2$. Such programs are structured in a declaration part and an instruction part. Sequential and parallel composition, iteration and non-deterministic choice are possible operations. Different processes can share common memory or use channel communication. The implementation of a procedure concept [8] and abstract data types extends $B(PN)^2$ to a complete programming language in which complex network protocols can be specified.

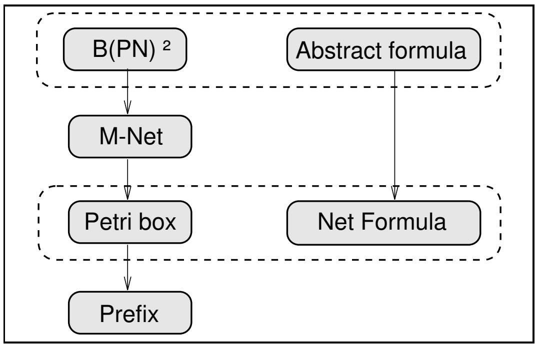

Figure 2.1: Dependencies between the objects of the **PEP** system.

Due to the fact that the semantics of a $B(PN)^2$ program is given in terms of Petri nets, theoretical results of this domain can be used for verification. As shown in figure 2.1, a $B(PN)^2$ program can be compiled in a compositional way into a Petri box, which is a special, 1-safe, labelled Petri net. M-nets, which give the high level net semantics of $B(PN)^2$ programs [1, 2] can be intermediate representations.

[e]provably Correct Communication Networks

During the verification phase it is necessary to calculate the finite prefix of a branching process. This prefix of a Petri net contains all the necessary information to check [8] various properties of Petri nets and can therefore be the input for an efficient partial order based model checking algorithm [6].

If the user wants to check whether a system satisfies a special property, (s)he can express this property in terms of a temporal logic formula. The model checking algorithm takes a formula over the Petri box as input. The transparency and thus the full functionality is reached by a formula transformator, which is able to transform an abstract formula into a formula over the Petri box.

3 Idea of the telecommunication protocol

The telephone network is modelled on the $B(PN)^2$ level by use of parallel processes. As shown in figure 3.1 there is one process for every telephone and a main part representing the network with the different possible calls. Three different processes participate if a connection from telephone "i" to "j" is going to be established. The part of the network corresponding to the "*call from i to j*" communicates with the "*calling part*" of telephone "i" as well as with the "*receiver part*" of telephone "j".

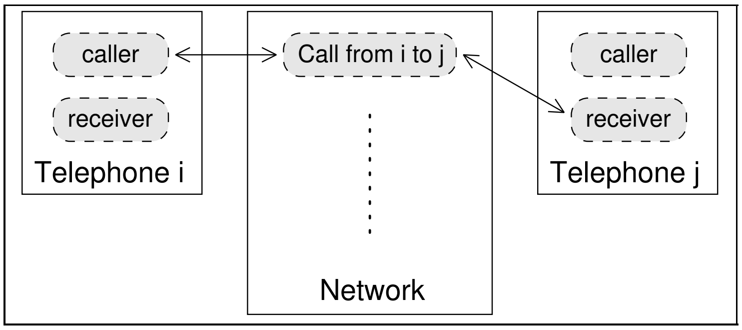

Figure 3.1 : Structure of the network.

4 Modelling in $B(PN)^2$

Due to the lack of space only a small part of the whole program can be shown here to give an impression of the way the modelling in $B(PN)^2$ can be done by use of procedures. In figure 4.1 a simple version of the "*receiving part*" of a telephone is depicted. The global channel (with capacity 0) "*teln*" is used for the communication related to telephone "n" (out of m). A telephone number to be called (ranging from 1 to m) or constants, e.g. "*alert*", "*no-sign*", "*off-hook*" or "*on-hook*" are sent and received (synchronously) via the channel. At any moment a telephone acts exclusively as a caller or receiver. Of course a telephone is modelled as an infinite loop, so that an infinite number of potentially different connections can be established during the run of the program.

proc *Telephone* (ref *teln*: chan 0 of $\{1..m$, *off-hook..abort*$\}$, n: $\{1..m\}$)
begin
do $\langle teln? = alert\rangle$;
  do $\langle teln! = no\text{-}sign\rangle$; exit
  □ $\langle teln! = off\text{-}hook\rangle$;   // point1: connection established
   do $\langle teln? = disconnect\text{-}tone\rangle$; $\langle teln! = on\text{-}hook\rangle$; exit
   □ $\langle teln? = abort\rangle$; $\langle teln! = on\text{-}hook\rangle$; exit
   □ $\langle teln? = on\text{-}hook\rangle$; exit   // action1: certain way of disconnection
   od; exit
  od; repeat   // point2: properly disconnected
   ⋮
od
end;

Figure 4.1: Part of a B(PN)^2 program

5 A corresponding Petri net

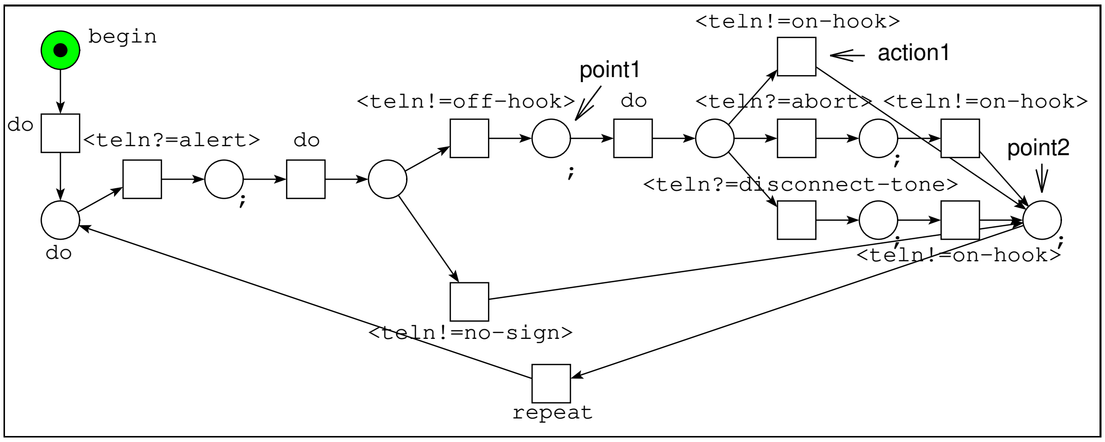

Figure 5.1: The corresponding automatically generated Petri net.

In this section the Petri net semantics of the small part of the B(PN)^2 program is introduced. The automatic Petri net generator keeps track of the references between transitions in the net and actions in the program as well as between places in the net and points in the control flow of the program. The transition labels shown in figure 5.1 are well-suited to understand the correspondence between the program and the net. It is even feasible to simulate the program by playing the token-game of the net. In figure 5.1 an isolated process is shown. In general a transition belongs to the control flow of more than one process and also to the local or global data flow. This is the main reason for the difficulties encountered during the verification of Petri net models for telecommunication protocols. The complexity of the verification of such protocols is largely independent from the formulation in B(PN)^2 or any other specification language.

6 Verification possibilities

The **PEP** system offers a variety of Petri net analyzing methods that allow the verification of various properties of a system modelled in B(PN)^2. Most of the algorithms implemented in the **PEP** tool are not based on the reachability graph. The so called "state explosion problem" is avoided by using a partial order based approach. The basis for the analysis is the finite prefix of the branching process, which is calculated automatically in an efficient way. Once this prefix is constructed, different properties and formulae can be checked by using the prefix. Furthermore all initially dead transition (together with the corresponding arcs) are eliminated during the calculation of the prefix.

First of all there are some standard algorithms like deadlock detection. But the main verification component is a model checking algorithm. Properties of 1-safe Petri nets, given in a linear or branching time logic can be checked. The atomic sentences of the logic are places. The usual boolean operators ($\neg$, $\wedge$ and $\vee$) are complemented by the modal operators $\Diamond$ (possibly) and $\Box$ (always). A formula can also argue about the maximal runs of the net by use of the quantifiers $\exists$ and $\forall$.

Furthermore the application of the model checker is not limited to Petri nets. A formula transformator allows the usage of macros (like "LIVE(action)[f]") on the one hand and the reasoning about actions in the program and points in the control flow on the other hand. The nesting of subformulae gives the full expressiveness of the logic.

Thus in our example a variety of different properties can be checked:

- Is a certain simultaneous combination of connections possible?
  $\Diamond$ (point1a $\wedge$ point1b)

- Is a certain way of disconnection always sometime in the future possible?
  LIVE (action1)

- Is a certain connection always properly disconnected?
  $\forall$ ($\Box$ (point1 $\Rightarrow$ $\Diamond$ point2))

- Is it possible that a certain connection is established an infinte number of times?
  $\Diamond$ (($\Box$ (point1 $\Rightarrow$ $\Diamond$ $\neg$ point1)) $\wedge$ ($\Box$ ($\neg$ point1 $\Rightarrow$ $\Diamond$ point1)))

During the modelling phase some typical human errors (like sending something on the channel – c! – instead of receiving – c? –) are likely to occur which can have severe effects. The **PEP** approach is well-suited to detect these mistakes. Thus for a faulty protocol that can neither be handled with a state graph based approach ($>>$ 1.000.000 states[g]) nor with a stubborn set reduced state graph based approach ($>$ 250.000 states[h]) the 4 properties were checked in less than one minute with the **PEP** tool (on a SPARC 10 with 40 MHz) after the finite prefix of the branching process had been calculated once within one hour. The deadlock detection was done in less than one second.

[f]It is expanded to the formula: $\Box$ $\Diamond$ (places in the preset of the transition belonging to the action).
[g]Probably more than 10.000.000 states
[h]The calculation had to be stopped after 2 days, because 100 MBytes of memory were used.

7 Conclusion

A brief introduction in the way telcommunication protocols can be modeled by use of the parallel programming language $B(PN)^2$ has been given. The advantage of the Petri net semantics of this language and the existence of an automatic Petri net generator has been stressed. Finally the various possibilities of the verification component of the **PEP** system for the analyses of telecommunication protocols has been reviewed. Thus it has been shown that the **PEP** tool is well-suited to model and to verify telecommunication protocols.

In the near future the implementation as well as the integration of an editor for finite automata and a compiler from a (parallel collection of) finite automata into a $B(PN)^2$ program or a Petri net will be finished.

References

[1] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets with an Application to the Semantics of $B(PN)^2$. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *LNCS*, pp. 103–118. Springer, 6 / 1995.

[2] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. A M-Net Semantics of $B(PN)^2$. In J. Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing. Springer, 1995.

[3] E. Best and R. P. Hopkins. $B(PN)^2$ – A Basic Petri Net Programming Notation. In A. Bode, M. Reeve, and G. Wolf, editors, *Proceedings of PARLE '93*, volume 694 of *LNCS*, pp. 379 – 390. Springer-Verlag, 1993.

[4] CCITT Recommendations Q.1200: Intelligent Networks, final version, WP XI/4, 3 / 1992.

[5] J. M. Duran and J. Visser. International Standards of Intelligent Networks. IEEE Communication Magazine, 2 / 1992.

[6] J. Esparza. *Model Checking Using Net Unfoldings*, pp. 151–195. Number 23 in Science of Computer Programming. ELSEVIER, 1994.

[7] J. J. Garrahan et al. Intelligent Network Overview. IEEE Communication Magazine, 3 / 1993.

[8] B. Grahlmann, S. Römer, T. Thielke, B. Graves, M. Damm, R. Riemann, L. Jenner, S. Melzer, and A. Gronewold. In E. Best and H. Fleischhack, editors, *PEP: Programming Environment Based on Petri Nets*, number 14/95 in Hildesheimer Informatik-Berichte. Universität Hildesheim, 5 / 1995.

[9] S. Kleuker. A Gentle Introduction to Specification Engineering Using a Case Study in Telecommunications. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *TAPSOFT'95*, LNCS. Springer-Verlag, 1995.