– PEP – More than a Petri Net Tool

Bernd Grahlmann and Eike Best*

ABSTRACT The **PEP** system (**P**rogramming **E**nvironment based on **P**etri Nets) supports the most important tasks of a good net tool, including HL and LL net editing and comfortable simulation facilities. In addition, these features are embedded in sophisticated programming and verification components. The programming component allows the user to design concurrent algorithms in an easy-to-use imperative language, and the **PEP** system then generates Petri nets from such programs. The **PEP** tool's comprehensive verification components allow a large range of properties of parallel systems to be checked efficiently on either programs or their corresponding nets. This includes user-defined properties specified by temporal logic formulae as well as specific properties for which dedicated algorithms are available. **PEP** has been implemented on Solaris 2.4, Sun OS 4.1.3 and Linux. Ftp-able versions are available.

KEYWORDS: $B(PN)^2$, Model checking, Parallel finite automata, **PEP**, Petri nets, Process algebra, Temporal logic, Tool.

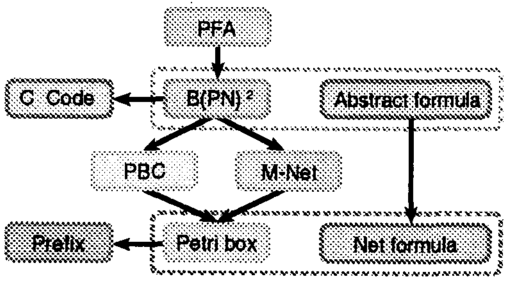

FIGURE 1. Objects used by the **PEP** system.

1 Introduction

In **PEP**[1] [5] two of today's most widely accepted theoretical approaches for the description of parallel systems, viz., Petri nets and process algebras, are combined to model, simulate, analyse and verify parallel systems. The integration of these two approaches, using a common, flexible parallel programming language called $B(PN)^2$ (**B**asic **P**etri **N**et **P**rogramming **N**otation), is one of the main characteristics of **PEP**.

*Institut für Informatik, Universität Hildesheim, Marienburger Platz 22, D-31141 Hildesheim, {bernd,E.Best}@informatik.uni-hildesheim.de

[1]The PEP project is financed by the DFG (German Research Foundation). The presentation of this paper has also been supported by the HCM Cooperation Network EXPRESS

This paper briefly describes the motivation and rationale behind the **PEP** tool [5]. Its structure and the interdependencies of its different components are explained. The different types of objects used by the **PEP** system are described. The efficiency of the model checking algorithms is demonstrated. An overview over future work and pointers to relevant literature are given.

2 Modelling parallel systems with **PEP**

Users can choose primarily between four types of objects in order to model parallel systems (see figure 1):

1. They can express parallel algorithms in $B(PN)^2$ [6], which is an imperative / predicative programming language whose atomic actions may contain predicates involving the prevalues and the postvalues of variables. Basic command connectives of $B(PN)^2$ are: sequential composition, nondeterministic choice, parallel composition, and iteration. Programs are structured into blocks consisting of a declaration part and an instruction part. Processes can share common memory or use channel communication or both. The implementation of a procedure concept [13] and abstract data types extend $B(PN)^2$ to a complete programming language. $B(PN)^2$ is called **B**asic **P**etri **N**et **P**rogramming **N**otation because it has a compositional semantics in terms of low-level (LL) Petri nets called boxes [1].
2. Terms of a process algebra called PBC (**P**etri **B**ox **C**alculus), which is an extension / modification of CCS [7], can be used. In **PEP**, PBC terms can either be derived automatically from a $B(PN)^2$ program or be designed independently.
3. a) **PEP** allows to design and edit arbitrary labelled P/T-nets. Boxes are a special case which may arise out of a translation from $B(PN)^2$ programs.
   b) Further, **PEP** allows to design and edit high-level (HL) nets, called M-nets [4], on which an alternative net semantics of $B(PN)^2$ programs is based [3].
4. Parallel finite automata (PFA) with $B(PN)^2$ actions as edge annotations can be edited and compiled into $B(PN)^2$ programs [10].

It is up to the user with which kind of object (s)he would like to start the modelling phase. As shown in figure 1, a $B(PN)^2$ program can be compiled into a Petri box in two different ways: using either PBC or a HL net as an intermediate step. The translations produce equivalent LL nets.
Further, the following objects are used in the **PEP** system:

1. In order to allow the user to (model) check a custom designed system property, **PEP** allows the definition of a set of temporal logic formulae in propositional logic on place names, augmented with $\Box$ for

'always' and $\Diamond$ for 'possibly at some future point'. Abstract formulae referring to points in the control flow, actions or variable bindings of a B(PN)^2 program are handled by an automatic transformation.

2. During verification it may become necessary to calculate the finite prefix of a branching process [8] of an existing LL net. This prefix contains information for model checking [9, 12] a net.
3. A C code generator can produce executable code [14].

In a modelling cycle several different objects are normally created. Typically a B(PN)^2 program is written, the corresponding HL net and the LL Petri box are compiled automatically, the prefix is calculated and interesting properties are expressed by formulae. All objects that relate to a single modelling cycle are collected into a machine object called project. A project which is the basic entity handled by the **PEP** system may also contain analysis and verification results or user defined documentation.

3 Structure of the PEP prototype

The **PEP** system consists of five different classes of components.

1. Editors for B(PN)^2 programs, PBC terms, Petri boxes (LL nets), M-nets (HL nets), formulae, PFA, C code and project documentation.
2. Compilers as follows: B(PN)^2 $\Rightarrow$ PBC, PBC $\Rightarrow$ (LL) Petri box, Petri box $\Rightarrow$ prefix, B(PN)^2 $\Rightarrow$ (HL) M-net, (HL) M-net $\Rightarrow$ (LL) Petri box, B(PN)^2 $\Rightarrow$ C code and PFA $\Rightarrow$ B(PN)^2.
3. Simulators for HL and LL boxes. The simulation of a B(PN)^2 program can be triggered by the simulation of a net, based on references either between actions of the B(PN)^2 program and transitions of the corresponding nets or between points in the control flow of the different processes and places of the nets.
4. Some standard algorithms (checking the free choice / T-system properties, liveness, deadlock-freeness, reachability and reversibility) are part of the **PEP** prototype.
5. Three model checking algorithms have been implemented, MC-0 for T-systems [17] and MC-1 and MC-2 [9, 12] for safe Petri nets. They can determine whether a Petri net satisfies a property given in terms of a temporal logic formula. The transparency, and thus the full functionality, of the verification component, is obtained by a formula translator which transforms an abstract formula (such as 'if a program terminates', 'if at some point a variable has a certain value' or 'if a transition is live') into a formula referring to a corresponding Petri net. This offers a comfortable interface to the model checkers.

The components are controlled by a project manager which represents the central part of **PEP** (see Figure 2).

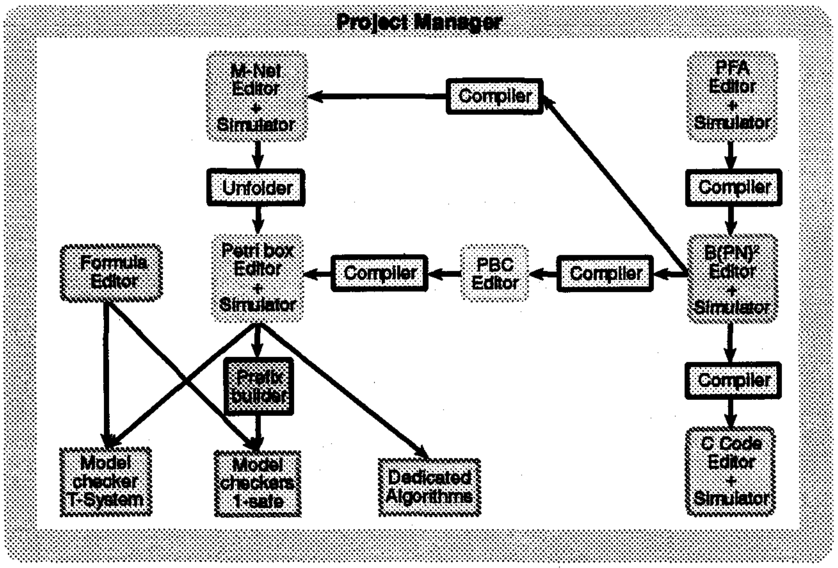

FIGURE 2. Interplay between **PEP** system components. The arrows represent input/output relations between components.

4 Efficiency of the verification component

During the work with the **PEP** system, different case studies have demonstrated that the tool is capable of handling comprehensive examples. As realistic examples a protocol for telephone networks [11] and a pilot behaviour model [15] can be mentioned.

The verification of faulty protocols showed that typical human errors made during the modelling phase can be detected efficiently [11]. Considering a protocol with much more than 1.000.000 states and more than 250.000 states (stubborn set reduced) the calculation of the finite prefix of the branching process was done in less than 1 hour, the verification of 4 interesting properties took 1 minute and the deadlock detection was done in 1 second.

W. Ruckdeschel and R. Onken [15] state '... *the* **PEP** *model checker is used to investigate large fusions of subnets characterized by 'exploding' reachability graphs. Some subsystems which could not be analysed via reachability graph were successfully unfolded by* **PEP** ...'

5 Future work

Our experiences have prompted us to consider the following future aims:

1. A model checker for HL Petri boxes should be developed and implemented, in such a way that the unfolding into LL Petri boxes – often yielding too big nets for realistic problems – can be avoided if desired.

2. The language $B(PN)^2$ should be augmented by the addition of abstract data types and by the full extension of the procedure concept. This improves the power to model complex parallel systems easily.
3. The possibilities of the INA system [16] (in particular, its state graph checker and its theorem data base) will be integrated.
4. 'SDL', 'VHDL' and 'OCCAM' are considered suitable for extending the interfaces of the tool.

6 References

[1] E. Best, R. Devillers and J. G. Hall. The Box Calculus: a New Causal Algebra with Multi-Label Communication. *Advances in Petri Nets 92*, *LNCS* Vol. 609, 21–69. Springer, 1992.

[2] E. Best and H. Fleischhack, editors. *PEP: Programming Environment Based on Petri Nets.* Hildesheimer Informatik-Berichte 14/95. 1995.

[3] E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. An M-Net Semantics of $B(PN)^2$. *Proc. of STRICT*, 85–100, Workshops in Computing. Springer, 1995.

[4] E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. *Proc. of ATPN'95, Torino*, *LNCS* Vol. 935, 103–118. Springer, 1995.

[5] E. Best and B. Grahlmann. *PEP: Documentation and User Guide.* Universität Hildesheim. ftp.informatik.uni-hildesheim.de /pub/Projekte/PEP/... or http://www.informatik.uni-hildesheim.de/~pep/HomePage.html 1995.

[6] E. Best and R. P. Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notation. *Proc. of PARLE*, *LNCS* Vol. 694, 379–390. Springer, 1993.

[7] E. Best and M. Koutny. A Refined View of the Box Calculus. *Proc. of ATPN'95, Torino*, *LNCS* Vol. 935, 103–118. Springer, 1995.

[8] J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Proc. of TACAS'96*, 1996.

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

[10] B. Grahlmann, M. Moeller, and U. Anhalt. A New Interface for the PEP Tool – Parallel Finite Automata. *Proc. of 2. Workshop Algorithmen und Werkzeuge für Petrinetze*, AIS 22, 21–26. FB Informatik Universität Oldenburg, 1995.

[11] B. Grahlmann. Verifying Telecommunication Protocols with PEP. *Proc. of RELECTRONIC'95, Budapest*, 251–256, 1995.

[12] B. Graves. *Ein Modelchecker für eine Linear-Time-Logik.* Diplomarbeit, Universität Hildesheim, 1995.

[13] L. Jenner. A Low-Level Net Semantics for $B(PN)^2$ with Procedures. In [2].

[14] S. Melzer. Design and Implementation of a C-Code Generator for $B(PN)^2$. In [2].

[15] W. Ruckdeschel and R. Onken. Petri Net Modelling, Analysis and Realtime-Simulation of Pilot Behaviour. *Proc. of ATPN'95*, 1995.

[16] P. H. Starke. *INA: Integrated Net Analyzer.* Handbuch, 1992.

[17] T. Thielke. Implementierung eines effizienten Modelchecking-Algorithmus. *Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*, 127–139. Springer, 1993.