APNNed - a Net Editor and Debugger within the APNN Toolbox

Peter Buchholz, Peter Kemper and the APNNed group*
Informatik IV
Universität Dortmund
D-44221 Dortmund, Germany
{buchholz,kemper}@ls4.informatik.uni-dortmund.de

Abstract

The paper presents APNNed, a new component within the APNN toolbox, that allows editing, visualization and debugging of certain colored Petri nets, which can result from a translation process of other Petri net tools. APNNed supports refinement of places and transitions to achieve hierarchical nets. Its facilities to perform the token game include consideration of break points as a feature well known from debugging software.

APNNed is the result of a students project, such that authorship is clearly shared with the 10 members of this group.

Keywords: Colored Petri nets, Hierarchical Petri nets, net editor

1 Introduction

The graphical representation is a natural advantage of Petri net formalisms, which has consequently lead to a large number of net editors for various classes of nets and in combination with various tools for their analysis.

Recent activities try to reduce the redundancy of formats, interfaces and formalisms. An ISO standardization process is under way to fix a formalism for colored untimed Petri nets as a first step and a full standardization later on. The current status is a draft[1]. There is the Petri net kernel development at Humboldt university to simplify the development of Petri net tools by providing an abstract data type and access function for nets [13, 12]. A current implementation with Python is presented in [12, 15] which offers programming interfaces to build up a specific tool with the help of a set of already available components. A more basic approach via a file format for a family of net classes has been presented in [2], the so-called "abstract Petri net notation" (APNN). Its main intention is to simplify

*Faisal Amir, Fatih Bosdag, Boris Goz, Stefan Illner, Marc Kaiser, Dimitri Knjazew, Selim Narci, Liliane C. Nouthe Sandjon, Dirk Pessarra, and Carsten Tepper

[1] For details see web page http://www.daimi.aau.dk/PetriNets/standard/

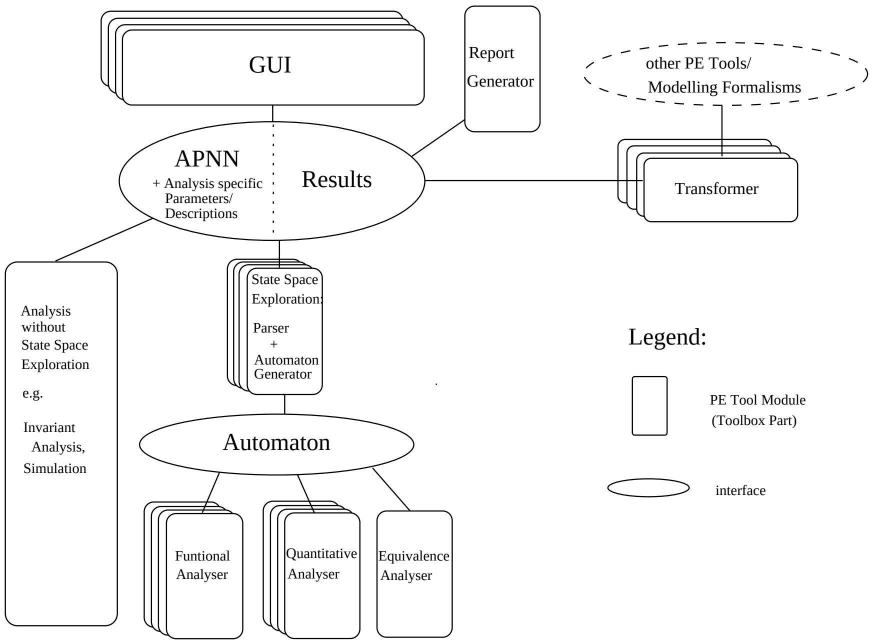

Figure 1: Structural overview of the toolbox

intertool communication, such that a separation of concerns is achieved between the development of graphical user interfaces and tool development for the analysis of nets, e.g. for simulation, reachability analysis, invariant analysis, model checking. A modular tool design around common interfaces reduces the effort of implementing new modules or tools drastically. A toolbox [1] has been developed around the APNN format for this purpose. Fig. 1 shows the different components and interfaces of the toolbox which include GUIs and several analysis programs. All cooperate via APNN, a textual interface, which is complemented by the description of analysis specific parameters, e.g., the definition of quantitative measures like rewards or functional requirements as CTL-formulas. The toolbox also comprises transformers, which translate other formats into the APNN. E.g., software translating GreatSPN [6] net specifications or PEP model descriptions [8] to the APNN. Once a model is given in APNN format, the toolbox provides a transformer from net level to the automata level describing the reachability graph of a net or subnet such that algorithms mainly for Petri nets can work with the APNN interface, while state based analysis can take place at the level of automata. The latter is a traditional focus point of research of our group at LS IV, especially analysis methods based on Kronecker representations of the underlying stochastic processes or transition systems have been developed in the recent years.

The APNN allows a separation of concerns, especially for classroom work: small projects can be defined, which concentrate on specific aspects, but the resulting modules can be integrated in the toolbox and then applied as one step in the analysis of complex models. An example for such a students project is the development of APNNed, which is a specific GUI to edit, visualize and debug hierarchical, colored, stochastic Petri nets.

2 Supported net classes of APNNed

APNNed considers colored Petri nets with an explicit enumeration of colors for places and transitions and with a classical refinement of places and transitions by substitution [9]. Additionally transitions may carry timing information as in stochastic Petri nets [5]. Every place has the following attributes: a name, a finite set of colors, and for each of its colors there is a name, the initial and current number of tokens (the latter is useful within the token game) and a screen color for visualization. Every transition has a name attribute and a finite set of colors which represent the different firing modes of the transition. Transitions also contain the incidence functions to define enabling and firing for each firing mode. A screen color is assigned to firing modes for visualization. Places or transitions can be refined. In this case, they become a graphical representation for a subnet, which itself can contain refined elements again. The coupling between subnet and supernet takes place via socket and port nodes adjacent to the refined node. APNNed as most Petri net editors does not allow for the refinement of adjacent nodes in order to avoid consistency problems and update anomalies. Fig. 2 shows a screen shot of a hierarchical example net where two transitions "proc1" and proc2" are refined. The window in the upper left of Fig. 2 indicates the hierarchical structure of the net similarly to the hierarchical structure of a directory tree.

3 Distinct features of APNNed

APNNed is implemented in JAVA, such that it inherits the (proposed) advantages of JAVA applications which include portability, ease of integration into interactive documents by a applets among others.

Clearly any net editor must include an essential functionality to be operational, which includes load and save, create and delete etc. APNNed can also load APNN descriptions of uncolored nets. In this case, the editor automatically transforms the uncolored net into a colored net, where all places and transitions have a color domain including a single color. Additional to this, automatic transformers have been developed which transform nets from other Petri net tools like PEP and GreatSPN into the APNN which can then be loaded with APNNed. With APNNed the nets can be restructured into hierarchical nets and the token game can be performed. In their current stage the transformers support only a subclass of nets from GreatSPN and PEP, namely extended Place/Transition nets. However, the extension of the programs to more general net classes is straightforward. The token game in APNNed is trace driven, which means that a trace is either loaded from a file, or generated by performing the token game step by step in an interactive mode or generated automatically with random selection of transitions for firing. Once a trace is available the user can move backwards and forwards within it, which is a useful feature to analyze traces generated by analysis tools to detect the reason for an observed behavior. Furthermore the automatic generation of a trace can be terminated by assigning breakpoints. A break point can be a lower or upper limit for the number of tokens on a place or the enabling of a transition. This feature allows to employ APNNed as a debugger, e.g. for nets which result from an

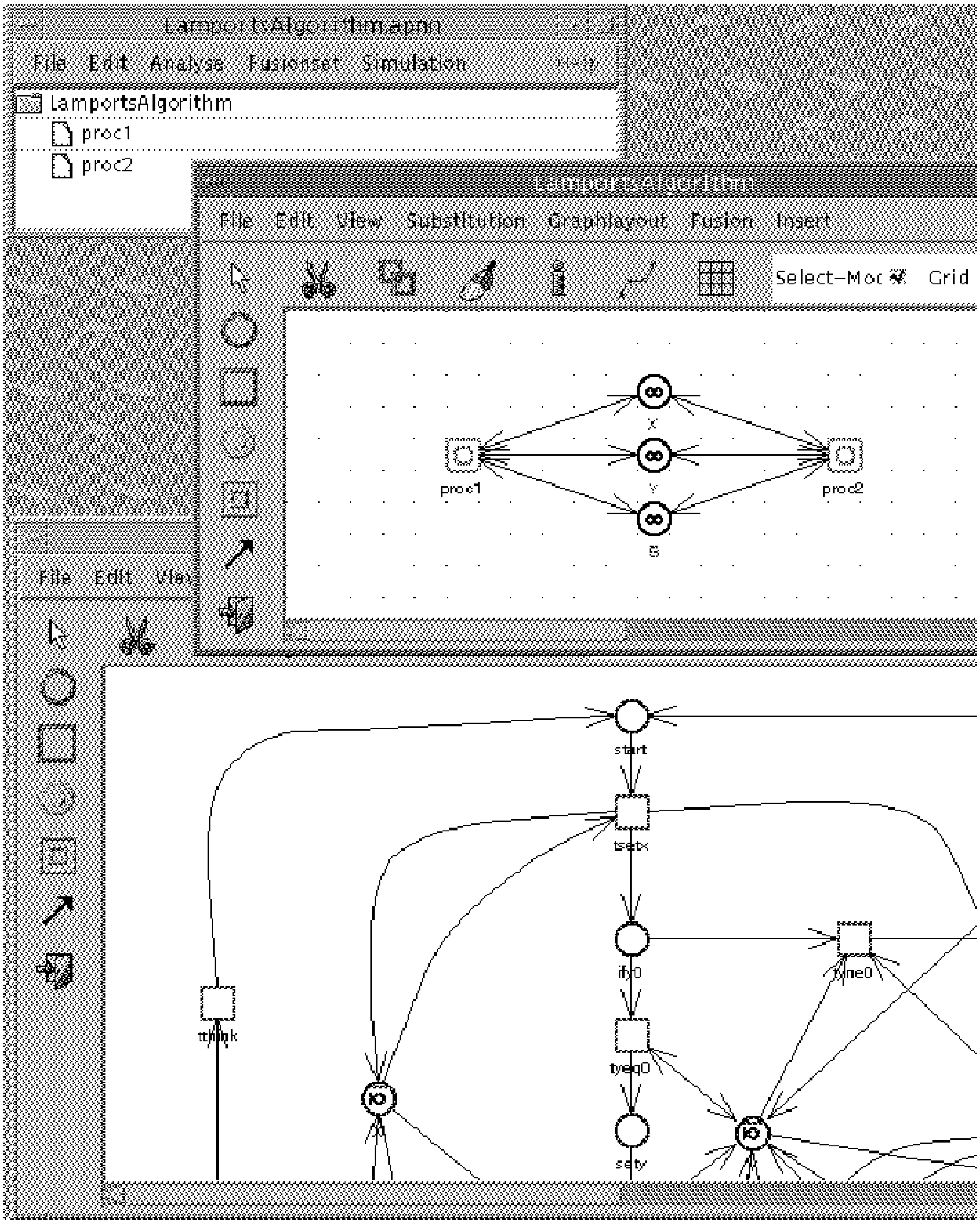

Figure 2: Lamports mutual exclusion algorithm as a hierarchical net

automatic translation process.

For such nets, APNNed includes an automatic layout function as well, which employs some heuristics to obtain a planar layout of a given net. The resulting layout is not guaranteed to meet the esthetic demands of a user but the feature can be helpful in many cases.

APNNed allows to manipulate the hierarchical structure not only by creation and deletion of subnets, but also by merging super- and subnets and by moving a selected set of nodes into a subnet. APNNed imports and exports nets in APNN format.

4 Integration in toolbox

A variety of tools have been developed around the APNN interface, these include rather different analysis tools, e.g.

invariant computation which is build upon an implementation of the classical Martinez/Silva algorithm [14] for Petri net analysis which originates from Farkas algorithm. This module can be used to obtain a generating set of minimal place or transition invariants, to test whether a net is covered by invariants, and to extend a net description by additional capacities obtained from place invariants for a given initial marking.

modelchecking for formulas in computational tree logic (CTL) and Petri nets which are superposed via fusion of transitions. The model checking algorithms as presented in [7] are applied within a Kronecker representation of the reachability graph and an ordered natural decision diagram (ONDD) like representation of the reachability set [11].

Markov chain analysis can be applied in two variants for superposed generalized stochastic Petri nets (SGSPNs): either by using a modular Kronecker representation of the associated continuous time Markov chain (CTMC) [10] or by a hierarchical representation using the algorithm of [3] or [4] to obtain a two level hierarchy.

At the current stage APNNed does only support subsequent analysis in terms of model specification and visualization but not in terms of control or result presentation, the latter is not difficult and subject to ongoing work.

5 Conclusion

We briefly presented APNNed, a net editor within the APNN toolbox, which supports hierarchical, colored and stochastic Petri nets. APNNed is the result of a students project to evaluate the strengths and weaknesses of JAVA for the development of graphical user interfaces for modelling environments of discrete event dynamic systems. The results achieved so far are encouraging, although JAVA's support for graphics is rather basic (which might become significantly better by the new release of JAVA foundation classes, but this seems to be a classical experience with software releases). As with any students project, there is still much work to be done in completing and extending the current status to make the editor more powerful in its visualization abilities and in its supported net formalisms.

Future work will integrate a visualization of the hierarchy generation algorithm of [4], further algorithms for an automatic layout, support in result representation and experimentation.

References

[1] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. Forschungsbericht 680, Fachbereich Informatik, Universität Dortmund (Germany), 1998.

[2] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletters*, 49:9–27, Oct 1995.

[3] P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop Petri Nets and Performance Models (PNPM'97), St-Malo (France), June 1997*, pages 81–90. IEEE CS Press, 1997.

[4] P. Buchholz and P. Kemper. On generating a hierarchy for GSPN analysis. *ACM Performance Evaluation Review*, 26(2):5–14, 1998.

[5] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: a definition at the net level and its implications. *IEEE Trans. Software Engineering*, 19(2):89–107, Feb 1993.

[6] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: a graphical editor and analyzer for timed and stochastic Petri nets. *Performance evaluation*, 24(1/2):47–68, 1995.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[8] Bernd Grahlmann. The PEP Tool. In *Proceedings of CAV'97 (Computer Aided Verification)*, volume 1254 of *Lecture Notes in Computer Science*, pages 440–443. Springer-Verlag, June 1997.

[9] P. Huber, K. Jensen, and R.M. Shapiro. Hierarchies in coloured petri nets. In G. Rozenberg, editor, *Advances in Petri Nets*, LNCS 483. Springer, Berlin, 1990.

[10] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.

[11] P. Kemper and R. Lübeck. Model checking based on kronecker algebra. submitted for publication, Nov. 1997.

[12] E. Kindler. Der Petrinetz-Kern: ein einfaches Anwendungsbeispiel. In J. Desel, A. Oberweis, and E. Kindler, editors, *7. Workshop Algorithmen und Werkzeuge für Petrinetze*, number 85 in Informatikberichte, Humboldt-Universität Berlin, 1997.

[13] E. Kindler and J. Desel. Der Traum vom universellen Petrinetz-Werkzeug - der Petrinetz-Kern. In J. Desel, A. Oberweis, and E. Kindler, editors, *3. Workshop Algorithmen und Werkzeuge für Petrinetze*, number 341 in Forschungsberichte, Institut AIFB, Universität Karlsruhe, October 1996.

[14] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.

[15] M. Weber. Der Petrinetz-Kern - eine Aufteilung in Invariantes und Variables. In J. Desel, A. Oberweis, and E. Kindler, editors, *7. Workshop Algorithmen und Werkzeuge für Petrinetze*, number 85 in Informatikberichte, Humboldt-Universität Berlin, 1997.