Mathematical Modelling (MATHMOD VIENNA '97), Wien, February 1997, ARGESIM Report No. 11, pp. 171-176.

HEINER, M.; POPOVA-ZEUGMANN, L. 1997b.
Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets, in E. Schnieder and D. Abel (eds.), Proceedings 5th EKA '97, Braunschweig, pp. 162-179. 1997.

HEINER, M. 1997c.
Verifaction and Optimization of Control Programs by Petri Nets without State Explosion; Proc. 2nd Int. Workshop on Manufacturing and Petri Nets held at Int. Conf. on Application and Theory of Petri Nets (ICATPN '97), Toulouse, June 1997, pp. 69-84.

HEINER, M.; POPOVA-ZEUGMANN, L. 1997d.
On Integration of Qualitative and Quantitative Analysis of Manufacturing Systems Using Petri Nets; Proc. 42. Int. wissenschaftliches Kolloquium (IWK '97), Ilmenau, September 1997, TU Ilmenau, Vol. 1, 557-562.

HEINER, M.; DEUSSEN, P.; SPRANGER, J. 1998.
A Case Study in Developing Control Software of Manufacturing Systems with Hierarchical Petri Nets; in Int. Journal of Advanced Manufacturing Technology, special issue on Petri Net Applications in Advanced Manufacturing Systems, to appear summer 1998.

KEMPER, P. 1997.
Superposition of Generalized Stochastic Petri Nets and its Impact on Performance Analysis; PhD Univ. Dortmund, Dep. of CS; Krehl Verlag 1997.

KEMPER, P.; LÜBECK, R. 1998.
Model Checking Based on Kronecker Algebra; private communication.

LAUTENBACH, K.; RIDDER, H. A. 1995.
Completion of the S-invariance Technique by Means of Fixed Point Algorithms; Fachbericht Informatik 10/95, Univ. Koblenz-Landau, 1995.

LEWERENTZ, C.; LINDNER, T. 1995.
Formal Development of Reactive Systems - Case Study Production, LNCS 891, Springer, 1995.

MCMILLAN, K. L. 1992a.
The SMV System, Techn. Report, Carnegie-Mellon Univ. 1992; http://www.cs.cmu.edu/~modelcheck/smv.html.

MCMILLAN, K. L. 1992b.
Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits; Proc. of the 4th Workshop on Computer Aided Verification, Montreal 1992, 164-174.

MCMILLAN, K. L. 1993.
Symbolic Model Checking, Kluwer Academic Publishers, Boston 1993.

MELZER, S.; ESPARZA, J. 1996.
Checking System Properties via Integer Programming; Proc. ESOP '96, Linköping, LNCS 1058, 250-264.

MOON, I. 1992.
Automatic Verification of Discrete Chemical Process Control Systems; PhD Carnegie Mellon Univ., Dep. of Chemical Engineering, August 1992.

MURATA, T. 1989.
Petri Nets: Properties, Analysis and Applications; Proc. of the IEEE 77(89)4, 541-580.

PETERSON, J. L. 1981.
Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, N.J., 1981.

PROBST, S. T. 1996.
Chemical Process Safety and Operability Analysis using Symbolic Model Checking, Ph. D. Thesis, Carnegie Mellon Univ., Dep. of Chemical Engeneering, 1996.

RAUSCH, M.; LÜDER; A.; HANISCH, H.-M. 1996.
Combined Synthesis of Locking and Sequential Controllers; Proc. WODES '96, Edinburgh, Aug. 1996, pp. 133-138.

REISIG, W. 1985.
Petri Nets; An Introduction; EATCS Monographs on Theoretical Computer Science Vol. 4, Springer 1985.

SEMENOV, A.; YAKOVLEV, A. 1996a.
Verification of Asynchronous Circuits Based on Time Petri Net Unfolding; Proc. 33rd ACM/IEEE Design Automation Conf. (DAC '96), Las Vegas, June 1996, 59-63.

SEMENOV, A.; YAKOVLEV, A. 1996b.
Contextual Net Unfolding and Asynchronous System Verification; Techn. Report 572, Univ. of NewCastle upon Tyne, Dep. of CS; Dec. 1996.

SPRANGER, J. 1997.
"FUNLite - A Parallel Petri Net Simulator", Proc. 42. Int. wissenschaftliches Kolloquium (IWK '97), Ilmenau, September 1997, TU Ilmenau, Vol. 1, 563 - 568.

STARKE, P. H. 1990.
Analysis of Petri Net Models (in German); Teubner, Stuttgart 1990.

STARKE, P. H.; ROCH, S. 1997.
INA - Integrated Net Analyzer Version 1.7, Manual (in German); Humboldt Univ. at Berlin, April 1997; http://www.informatik.hu-berlin.de/~starke/ina.html.

TIEDEMANN, R. 1997.
PED - Hierarchical Petri Net Editor, Manual (in German); BTU Cottbus, Dep. of CS, Techn. Report, May 1997; http://www-dssz.Informatik.TU-Cottbus.De/~wwwdssz/ped.html.

VALMARI, A. 1992a.
Alleviating State Explosion during Verification of Behavioral Equivalence; Univ. of Helsinki, Department of Computer Science, Report A-1992-4, Helsinki 1992; ftp://saturn.hut.fi/pub/reports.

VALMARI, A. 1992b.
A Stubborn Attack on State Explosion; Formal Methods in System Design 1(1992)4, 297-322.

VARPAANIEMI, K. 1994a.
On Computing Symmetries and Stubborn Sets; Helsinki Univ. of Technology, Digital Systems Laboratory, Series B, Report No. 12, Espoo 1994.

VARPAANIEMI, K. 1994b.
On Combining the Stubborn Set Method with the Sleep Set Method, LNCS 815, Springer 1994, 548-567.

VARPAANIEMI, K.; HALME, J.; HIEKKANEN, K.; PYSSYSLAO, T. 1995.
PROD Reference Manual, Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, Espoo, August 1995; ftp://saturn.hut.fi/pub/reports.

WIMMEL, G. 1997.
A BDD-based Model Checker for the PEP Tool; Univ. of Newcastle, Dep. of CS, Major Individual Project, May 1997.

WIKARKSI, D.; HEINER, M. 1995.
On the Application of Markovian Object Nets to Integrated Qualitative and Quantitative Software Analysis, ISST-Berichte 29/95, Fraunhofer ISST, Berlin, Oct. 1995.

mh@informatik.tu-cottbus.de

cally by help of general Petri net analysis tools. Therefore, they are reproducible in an objective way.

For a general framework for Petri net based development and analysis of dependable systems, we conclude the following design criteria. At first, dedicated technical languages are needed to express functional, safety, and performance requirements as well. Second, the framework has to be customizable. Its components (editors, analysis tools, simulation tools, code generation facilities) should be interchangeable. For a given configuration, user guidelines are required showing which analysis techniques are recommendable in which order for a given analysis question. Additionally, design criteria are required which promotes meaningful analyses at each phase of development.

For specific application areas, dedicated configurations of the framework can be defined involving also an adaptation of the libraries and the terminology of the user interface. For instance, in manufacturing control in general, it seems to be possible to compile Petri net libraries of

- patterns which describe the communication structure of certain devices on a cooperation level (for our production cell case study , three such patterns are identifiable, each of them applicable to at least two devices),
- patterns which are suitable to describe elementary motion steps of the devices, and
- the associated environment models.

Using these libraries, it will be possible to develop control programs for the supported types of manufacturing systems by composition and refinement of instantiated net patterns.

In particular, in case of programmable logic controllers, the tool box's user interface may be adapted to the notions of the IEC 1131-3 standard.

REFERENCES

BALBO, G. 1992.
Performance Issues in Parallel Programming; LNCS 616, Springer, 1992, 1-23.

BEN-ARI, M.; PNUELI, M. A.; MANNA, Z. 1983.
The Temporal Logic of Branching Time; Acta Informatica 20(83), 207-226.

BEST, E.; GRAHLMANN, B. 1996.
PEP - more than a Petri Net Tool; Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), Passau, March 1996, LNCS 1055, Springer 1996, 397-401; The system is available on ftp.informatiuni-hildesheim.de.

BEST, E. 1996.
Partial Order Verification with PEP; in Holzmann, G.; Peled, D., Pratt, V. (eds), Proc. POMIV '96, Princeton, Am. Math. Soc. 1996, 305-328.

BRYANT, R. E. 1986.
Graph-based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers C-35(86)8, 677-691.

BRYANT, R. E. 1992.
Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams; ACM Computing Survey 24(1992)3, 293-318.

BUCHHOLZ, P. 1991.
The Structured Analysis of Markovian Models (in German); Informatik-Fachberichte 282, Springer 1991.

CLARKE, E. M.; EMERSON, E.A.; SISTLA, A. 1986.
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications; ACM Trans. on Programming Lnguages and Systems 8(86)2, 244-263.

CORBETT, J. C. 1994.
Evaluating Deadlock Detection Methods for Concurrent Software; Proc. Int. Symposium on Software Testing and Analysis, 1994, 204-215; extended version in Techn. Report, Dep. of Information and CS, Univ. of Hawaii at Manoa, 1995.

COURCOUBETIS, C.; VARDI, M. Y.; WOLPER, P.; YANNAKAKIS, M. 1992.
Memory Efficient Algorithms for the Verification of Temporal Properties; Formal Methods in System Design 1(1992)2/3, 275-288.

DESEL, J.; ESPARZA, J. 1995.
Free Choice Petri Nets; Cambridge Tracts in Theoretical Computer Science 40, Cambridge Univ. Press 1995.

EMERSON, E. A. 1990.
Temporal and Modal Logic; in Leeuwen, J. v. (ed.), Handbook of Theoretical Computer Science, Vol. B; Elsivier, Amsterdam 1990, 995-1072.

ENGELFRIET, J. 1991.
Branching Processes of Petri Nets; Acta Informatica 28(1991), 575-591.

ESPARZA, J. 1993.
Model Checking Using Net Unfoldings; Proc. TAPSOFT '93, LNCS 668, Springer 1993, 613-628; full version in Science of Computer Programming 23(1994), 151-195.

ESPARZA, J.; RÖMER, S.; VOGLER, W. 1996.
An Improvement of McMillan's Unfoldig Algorithm; Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), Passau, March 1996, LNCS 1055, Springer 1996.

ESPARZA, J.; MELZER, S. 1998.
Verification of Safety Properties Using Integer Programming: Beyond the State Equation; private communication.

GERMAN, R. ET AL. 1994.
TimeNet - A Tool Kit for Evaluating Non-Markovian Stochastic Petri Nets, Techn. Univ. Berlin, Dep. of Computer Science, Report 1994-19; http://pdv.cs.tu-berlin.de/forschung/timenet.html.

GERTH, R., PELED, D., VARDI, M. Y., WOLPER, P. 1995.
Simple On-the-fly Automatic Verification of Linear Temporal Logic; Proc. 15th International Symposium on Protocol Specification, Testing and Verification (PSTV'95), Warsaw 1995, 3-18.

GODEFROID, P. 1996.
Partial-Order Methods for the Verification of concurrent Systems; LNCS 1032, 1996.

HEINER, M. 1992.
Petri Net Based Software Validation, Prospects and Limitations, ICSI-TR-92-022, Berkeley/CA, 1992.

HEINER, M.; VENTRE, G.; WIKARSKI, D. 1994.
"A Petri Net Based Methodology to Integrate Qualitative and Quantitative Analysis", J. Information and Software Technology, 36(7), pp. 435-441, 1994.

HEINER, M.; DEUSSEN, P. 1995.
Petri Net Based Qualitative Analysis - a Case Study; Brandenburg Techn. Univ of Cottbus, Dep. of Computer Science, Techn. Report I-08/1995; http://www-dssz.Informatik.TU-Cottbus.DE/~wwwdssz.

HEINER, M. 1997a.
On Exploiting the Analysis Power of Petri Nets for the Validation of Discrete Event Systems; Proc. 2nd IMACS Symposium on

That's why these transitions are superfluous in the given case, and we are able to optimize our model by deleting them. For more details see (Heiner 1997c).

We get an optimized model with the same state space as the unoptimized one, but without far less dynamic conflicts. For this version, the liveness for each transition of the considered pusher chains has been proven by model checking the corresponding temporal formula based on the branching process' prefix.

6.2 Special Analysis

Special analysis deals with properties reflecting the intended special functionality. The verification of safety properties is here especially important. Therefore we will stress this topic in the following paragraph.

(a) safety

There are different analysis techniques available to prove the unreachability of unsafe states:

Facts (INA): The unsafe states may be modelled as facts (special transitions which are expected to become never enabled). But, the evaluation of bad states (a state where a fact is enabled) by the given tool box requires the reachability graph. That's why we will avoid this approach.

Stubborn set reduction (INA): The net is transformed in such a way that the unsafe states become dead states. Then the stubborn set reduced reachability graph has to be constructed. Because any dead states are preserved under this reduction, the original net does not contain any unsafe states if the transformed net does not reach any dead states. This technique could be useful if the required net transformation is realized by the analysis tool.

Place invariants (INA): A sufficient condition for the unreachability of a given marking m is fulfilled if there exists at least one place invariant x for which the token conservation equation

$$\sum_{p \in P} x(p) \cdot m_0(p) = \sum_{p \in P} x(p) \cdot m(p)$$

is not valid.

State equation (INA): Let C denote the $P \times T$ integer incidence matrix of the given Petri net, and $\wp(q)$ the Parikh vector of a firing sequence $q \in T^*$. Then the following linear programming problem, called the state equation

$$m = m_0 + C \cdot x \ , \quad x \geq 0$$

has for each reachable marking at least one solution, namely $x := \wp(q)$. So the feasibility of the state equation is a necessary condition for the reachability of the marking m, or the infeasibility is a sufficient condition for the unreachability of m.

To check these both equations, complete markings must be specified. But unsafe states are usually given in terms of submarkings (containing "don't care" places). This main disadvantage is overcome in the next approach.

Trap equation (PEP): Based on a linear upper approximation of the state space, we have a sufficient condition for linear properties of the type $A \cdot m \leq b$ The implementation is going to be integrated in an improved version of PEP.

Model checking of temporal formulae: Model checking, combined with stubborn set reduction (PROD, LTL\X) or based on the finite complete prefix of a branching process (PEP, CTL_0), provides generally the most convenient method to raise safety questions, esp. because set of (unsafe) states may be characterized in a concise manner. Both model checkers run very fast. Due to the evaluation method, they are applicable also to larger systems of which the size of the interleaving state space is unknown.

(b) progress

Progress properties use in general a richer set of (temporal) logical operators. Therefore, model checking facilities are unavoidable. Due to the **AG** and **AF** operators, these properties can be proven only by those tools providing the whole CTL or LTL.

(c) consistency

General consistency conditions are analyzable by INA, and in the temporal logic version by PROD or PEP. But for larger systems, it is generally a cumbersome task to prove this type of properties by finding the suitable place invariants.

7 CONCLUSIONS

According our experience gained up to now, at least for the analysis of a restricted class of concurrent systems modelled by Petri nets, the construction of the complete state space can be avoided by a suitable combination of different methods (possibly implemented by different tools). This class can be characterized as follows:

- (Intentionally) life and 1-bounded systems (hence, covered by semipositive place and transition invariants),
- a certain degree of concurrency (which increases the efficiency of partial order methods and partial order representation methods),
- moderate amount of dynamic conflicts (non-determinism).

So, all qualitative (i.e. timeless) properties of our case studies have been proven without construction of the complete reachability graph (interleaving state space). Up to now, the quantitative (i.e. time-dependent) analysis of interval nets is based on reachability graph construction and evaluation. But in (Semenov and Yakovlev 1996a), a method has been proposed to describe the behaviour of interval nets by a finite prefix of the branching process. It seems to be worth thinking over how to combine both approaches. Nevertheless, all proves were carried out automati-

mh@informatik.tu-cottbus.de

Each physical device is basically characterized by its finite set of discrete states (maybe representing equivalence classes of possibly infinite sets of states), and additionally by the commands (externally visible transitions) forcing the device to change its current state. Obviously, each device must be in one and only one state at any time. In terms of Petri net theory, the states of a device form a place invariant establishing a consistency condition of the system model. In our concurrent pushers example, there are two types of devices (relays, pushers), while in the production cell five different ones exist. Accordingly, there are different consistency conditions to check during special analysis.

Control Program Model. In case of the pusher example, the original programmable logic controller programs are written in IEC 1131-3. General transformation rules have been introduced to transform automatically programmable logic controller programs into ordinary place/transition Petri nets. The automatization of this translation is part of a running project. By this way, the rich amount of available Petri net analysis techniques and tools become applicable for computer-aided analysis of programmable logic controller programs. For an example, how this could look like, see (Heiner 97a).

In order to avoid unnecessary restrictions of the concurrency degree, it could be helpful to exploit a special test arc feature for modelling of the transitions' side conditions. In that case, the amount of data, which has to be searched through during the analysis steps, may become much smaller, provided the analysis tools are prepared to handle test arcs (Semenov and Yakovlev 1996b).

An overview of the corresponding basic modelling steps to develop step-by-step the total system model is given in figure 4.

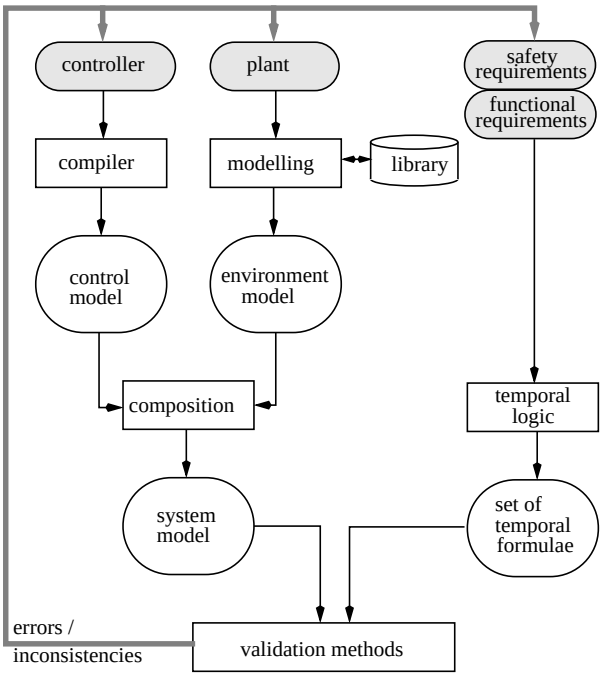

Figure 5: Basic modelling steps.

6 ANALYSIS

In the following we summarize our experience gained up to now while analysing the mentioned case studies (see section 4) by our tool box (see section 3).

6.1 General Analysis

General analysis deals with properties which should be valid independently of the intended functional behaviour of the system. Basically, these are boundedness and liveness, putting up together the well-formedness criteria.

boundedness: All nets are covered by semi-positive place invariants. This proof lasts several seconds in the worst case if taken by INA. Moreover, the token sum of all these place invariants equals to 1. So we are able to conclude the 1-boundedness of the net (a necessary precondition for PEP's model checker).

liveness: The deadlock freedom of all nets can be proven very efficiently by construction of stubborn set reduced reachability graphs (INA, PROD), which are in all processed examples much smaller than the complete state space.

Additionally, it can be shown efficiently that the net is covered by semi-positive transition invariants as necessary (but not sufficient) condition for liveness. Moreover, the deadlock-trap property helps to proof the liveness of the production cell's cooperation models.

But liveness (no dead system parts) can't be proven by classical Petri net theory neither for the production cell's control model nor for longer pusher chains. Reasons are the lack of suitable net structures (the given nets are not extended simple ones, therefore the deadlock trap property could not help; the known net reduction rules do not work;), and the well-expected state explosion by considering all interleaving transition sequences (reachability graph). We get state spaces up to 10^7 for the control model of the production cell, and up to 10^9 for 10 concurrent pushers chained up.

The way-out could be a liveness proof for each transition by model checking the temporal formula: **AG EF** $(en(t))$ based on the branching process' prefix (here en(t) stands for the enabling condition of t). This idea works for all versions of the production cell.

However, the prefixes are also unconstructable for more than 6 pushers. The reason for that seems to be the "useless" dynamic conflicts caused by transitions reproducing the current state in an active way. Test arcs to model side conditions might be helpful here. These dangerous transitions are part of general net components for context-independent modelling of basic statements. But in case of the given Petri net, modelling a programmable logic control program, these basic statements appear (only) as side conditions of the control flow, and never within the control flow.

describes only the synchronization of the machine controllers, while the refined control model includes also the interactions of the controllers with the controlled plant. The complete hierarchical model (with the size of about 200 places and 200 transitions structured into 65 pages) together with analysis protocols are published in (Heiner and Deussen 1995), a shorter and updated (by recent analysis results) version is going to appear in (Heiner et al. 1998).

The **concurrent pushers** are an adopted version of the pusher problem for which in (Rausch et al. 1996) a control program has been synthesized automatically. By this way, our verification presents a reversal check for that synthesis. The example consists basically of two concurrently working pushers moving work pieces (see figure 3). The work piece is moved from position one to position two by the first pusher, and from position two to position three by the second pusher. Both pushers are driven by electric motors which can be controlled by corresponding relays into two moving directions. Starting from this basic situation, chains of concurrent pushers may be constructed in order to move pieces step by step from the input position via a number of inner positions to the output position.

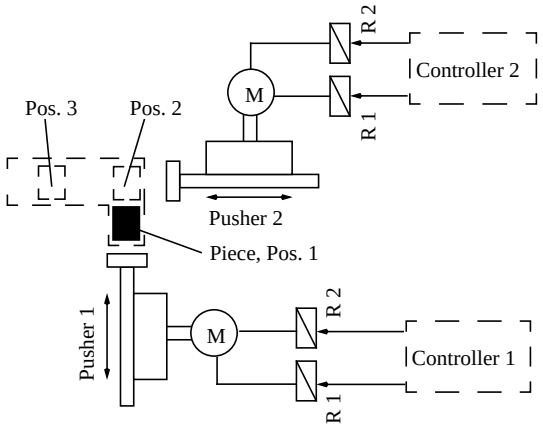

Figure 3: Concurrent pushers.

A characterization of the scalable model (chains up to 10 pushers have been considered resulting into net size of about 190 places and 215 transitions), and the essential analysis results are published in (Heiner 1997a). In (Heiner 1997c) a more detailed discussion can be found concerning two modelling versions and related consequences with regard to the analysis expenses involved.

Currently we are working on case studies taken from (Moon 1992), and (Probst 1996) to get a deeper insight into the pros and cons of OBDD-based methods in comparison with other techniques.

Moreover, the interested reader is referred to (Corbett 1994) for a related experience report. There, a tool combination covering integer programming (INCA), partial order reduction (SPIN) and OBDD (SMV) has been used for deadlock analysis of about 20 examples of different sizes. The efforts to analyse the same examples with PEP's prefix builder and model checker are reported in (Best 1996).

Requirement Specification. In addition to the task descriptions, more or less exhaustive lists of usually informally specified functional and safety requirements are given. Typical properties of this type are in case of the concurrent pushers e.g.:

(a) safety

- At any time, a pusher can be driven in one direction only.
- To avoid collisions, it is not allowed to move adjacent pushers at the same time.
- No pusher motion must be driven too far/near.
- While moving a pusher, a new work piece must not arrive in its input position.

(b) progress

- After an active phase of a pusher, its successor will be activated before the predecessor will be started again.
- It is guaranteed that each pusher works infinitely often (livelock freedom).
- Any work piece entering the plant will finally leave the plant.

(c) consistency

- Additional properties to be verified emerge during modelling reflecting useful (self-) consistency checks (see section 5).

5 Modelling with Hierarchical Petri Nets

The models of the total systems may be characterized by a strong separation of controller software and environment into different parts. A controller program consists generally of a finite and static set of communicating processes. The environment models are composed of small reusable components: the producer/consumer processes of the work flow, and the devices of the controlled plant.

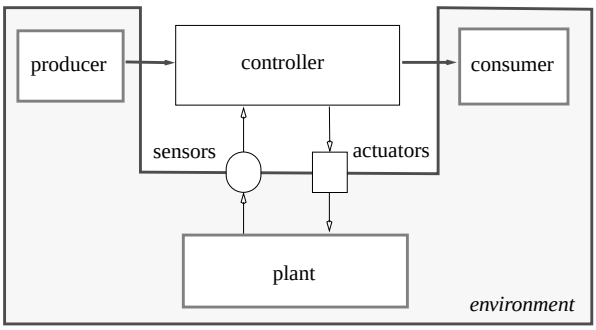

Figure 4: General system model structure.

Environment Model. There exist net components for each device type - building step-by-step a growing reusable component library to describe the uncontrolled plant behaviour.

mh@informatik.tu-cottbus.de

Table 1: Temporal logics comparison.

Tool	Supported type of logic	Types of properties
INA1.7	**EF** φ (but φ can only be given by a (sub-) marking)	(1), (2)
INA2, PROD/ PROBE, SMV	(full) CTL, i.e. **EX**/**AX**, **EF**/**AF**, **EG**/**AG**, **EU**/**AU**	(1) - (5)
PEP	CTL_0: **AG**, **EF**	(1) - (4)
PROD/ on-the-fly	LTL\X: (without nexttime operator) **G**, **F**, **U** (unquantorized versions of **AG**, **AF**, **AU**)	(2), (3), (5)

quite lengthy session protocols. Many model checkers (PROD and SMV, but not PEP) produce an execution path (sequence of states) on which a property in consideration is violated or a trace (transition sequence) to a state which violates this property, depending on whether LTL or CTL is provided.

Which tools *should* be applied in which order depends on the analysis methods they are based on. Which tools *may* be applied at all for a given type of question depends on their power to express a specific analysis question. To highlight differences between the applied tools concerning their expressiveness, it is useful to summarize typical questions/properties dealt with during analysis. In the following φ stands shortly for a general logical expression characterizing usually a (wanted or unwanted) state or set of states.

(1) **reachability-related** properties of the logical form **EF** φ : Reachability of a state where φ holds; there exists at least one computation path (future behaviour) to reach eventually a state where φ will be true.

(2) **safety-related**, **AG** (¬φ) or equivalent ¬**EF** φ : Unreachability of a state where φ holds; for every computation path, φ will never be true.

(3) **invariant-related**, **AG** φ or equivalent ¬**EF** (¬φ) : General validity of an assertion φ; for every computation path, φ will be true for ever.

(4) **liveness-related**, **AG EF** φ : What ever happens, there exists the chance (at least one path) that φ will be true.

(5) **progress-related**, **AG AF** φ : For every computation path, φ will eventually be true.

Table 1 describes which tool may be applied for which type of logical expression. The need to combine a variety of analysis tools stems from the different features (to raise different questions) or different analysis methods (to answer similar questions in a different way) they provide. Each of these tools has its strength and limits. So, they do not compete, but complement each other. The decision, which kind of analysis methods in which order is advisable and leads to results most efficiently, seems to depend generally on the application area.

4 CASE STUDIES

A search trough the literature reveals a lot of examples which are "a must" while performing methods/tools comparisons. Besides the well-known set of low-level mutual exclusion algorithms, such famous academic examples like Dijkstra's dining philosophers, and Milner's schedulers have been processed by our tool box. The mutex algorithms seem to be an unavoidable exercise to learn which properties are analyzable in which manner. But their very small state spaces make them uninteresting for efficiency comparisons. In opposite to that, the scalable philosophers and schedulers allow quick test series with increasing state spaces to get a first feeling for the practicability limits of current tool implementations. Unfortunately, many practical situations are rather unlikely to exhibit such regular structures, which makes such conclusions based on academic test series of limited value.

Therefore, the focus of our investigations builds real-life case studies of realistic sizes. Let's give two examples - one for each approach of Petri net based software engineering mentioned in section 2. For the first example *production cell*, a Petri net specification has been developed and validated a priori, while in the second example *concurrent pushers* the starting point was a programmable logic controller written in IEC 1131-3, for which a posteriori a Petri net has been derived.

The **production cell**, a really existing industrial facility, comprises six physical components: two conveyor belts, a rotatable robot equipped with two extendable arms, an elevating rotary table, a press, and a travelling crane. The machines are organized in a (closed) pipeline (figure 2). Their common goal is the transport and processing of metal plates. Altogether there exist 14 sensors and 34 actuators to control the cell. For more details of the task description the reader is referred to (Lewerentz and Lindner 1995). In that book, also a list of safety and liveness requirements can be found which are to obey by an implementation of the control program.

We have developed and analysed the control software stepwise in two abstraction levels constituting the cooperation model and the control model. The more abstract cooperation model

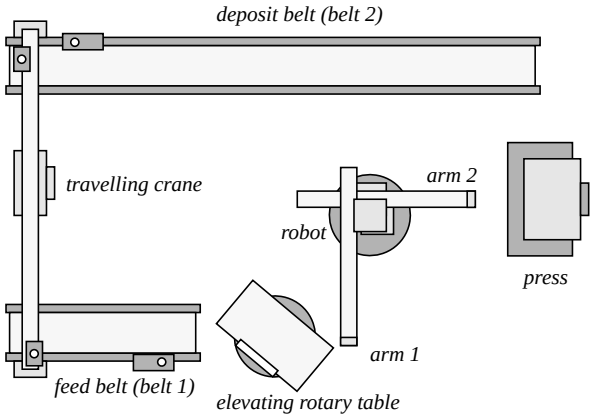

Figure 2: Top view of the production cell.

explicit state by state enumeration of the state space.

(3.2) Lazy state space constructions build reduced state spaces instead of the complete ones. Famous and well-known examples are the coverability graph, a finite representation of the infinite state set of unbounded systems, and the symmetrical reduced reachability graph exploiting state symmetries. Both methods have in common that each set of states, which are regarded to be equivalent under the given equivalence notion, is represented by only one state.

Moreover, if we deal with system properties which are invariant under the interchanging of concurrent transition occurrences, it is superfluous to consider all those interleavings separately. It is only necessary to construct a **reduced version** of the interleaving state space of the system, which is generally much smaller than the complete state space, especially in case of highly concurrent systems. In this way, e. g. all dead states can be found, or the un-/reachability of special states can be decided. Methods based on this idea are sometimes called **partial order methods** (a term which should be sharply distinguished from partial order representation techniques described below). A well-tried example of a partial order method is Valmari's *stubborn set* approach (Valmari 1992a), which can be combined with other techniques like the *sleep set method* (Godefroid 1996), (Varpaaniemi 1994b) and the *symmetry method* (Varpaaniemi 1994a). Valmari developed a generalization of his method in a way that properties expressible by Linear Time Temporal Logic (LTL) without the nexttime operator **X** are preserved by the reduction process (Valmari 1992b). Therefore, the standard model checking technique for LTL (Courcoubetis et al. 1992), (Gerth et al. 1995) can be applied to stubborn reduced reachability graphs.

(3.3) An alternative class of approaches to handle the state explosion problem bases on **partial order representations** of the behaviour of a concurrent system. Instead of sequences of events (i. e. occurrences of transitions), partially ordered sets of events are used as behaviour description. Partial orders of events can be interpreted in the following way: If an event precedes another one then the former one causes the latter, or the former one has to occur earlier in time than the latter. Since state explosion is in general caused by the representation of all possible interleavings of concurrent actions, partial order representations tend to be much smaller than reachability graphs, at least in case of a moderate amount of non-determinism.

A currently intensively discussed partial order representation approach is the construction of a "finite complete prefix of a branching process" of a Petri net (shortly called the *prefix* of the net) (Engelfriet 1991), (McMillan 1992b), (Esparza 1993). An further improvement has been introduced in (Esparza et al. 1996). The possible behaviour of a safe (1-bounded) net is represented by another, so-called occurrence net. Based on this net prefix, efficient model checking is possible for a very restricted subset of CTL comprising only the temporal operators **AG** and **EF.**

The tool box currently used (compare figure 1) is as follows. The Petri net EDitor PED with its hierarchy browser (Tiedemann 1997) supports basically the construction of hierarchical place/ transition nets. Complementary, all necessary attributes (esp. time attributes) of those net types can be assigned to appropriate net elements, which are analyzable by the evaluation tools linked up:

- PEDVisor allows to animate the functional behaviour by playing the token game.
- INA, version 1.7 (Starke and Roch 1997) provides an almost complete set of the (currently) known static and dynamic analysis techniques of "classical" Petri net theory. Additionally, its analysis methods of time-dependent (duration and interval) Petri nets have been applied extensively. Since version 2, a CTL model checker running on the reachability graph is also provided.

The next three tools follow the model checking approach, using (different versions of) propositional temporal logics as a flexible query language for asking questions about the (complete/ reduced) set of reachable states. By this way, even very large state spaces become manageable. But, the state space has to be finite for that purpose. So, boundedness is here an unavoidable precondition.

- PROD (Varpaaniemi et al. 1995) supports CTL, based on the reachability graph, as well as LTL, using on-the-fly stubborn set reduction.
- PEP (Best and Grahlmann 1995) offers, besides many other interesting things not used here, a model checker for a restricted type of CTL based on a partial order representation of the system behaviour. Its application is however restricted to 1-bounded nets.
- SMV (McMillan 1992a, 1993) provides a model checking technique for CTL based on OBDD's. This tool tailored to hardware verification has been linked up based on the ideas outlined in (Wimmel 1997). The structure of a Petri net is translated into the SMV's input language by defining a Boolean expression which computes the set of all possible successor states for a given state. This approach turns out to be rather inefficient, since SMV is unable to determine dependencies between place variables which could be used to compute smaller OBDD's.
  Therefore, we are going to realize a dedicated OBDD implementation for CTL model checking of the state spaces of Petri nets. First computational results are very encouraging.
- TimeNet (German 1994) supports the evaluation of generalized stochastic Petri nets by simulation as well as by analysis based on Markovian processes.
- FUNlite (Spranger 1997) allows the generation and (token-driven) distributed processing of executable code.

Information about the results of the analyses are recorded in several protocols. The type of such information depends on the analysis method and the tool used. To give a few examples, in general analysis e.g. dead states or upper bounds for the number of tokens located at some place (if any) are recorded by INA in

mh@informatik.tu-cottbus.de

- the integration of qualitative as well as quantitative analysis on the basis of a common representation of the system under development.

3 TECHNIQUES AND TOOLS

Concerning the methods to validate Petri nets (semi-) automatically, three different types can be basically distinguished: *animation* (1), *static* (2) and *dynamic* techniques (3).

(1) Net-based **animation** aims at functional behaviour simulation by playing the token game. The main advantage consists in a deeper insight into the net and - therefore at the same time - into the system behaviour. The results gained depend on the abstraction level of the underlying net model. But in any case, this special version of prototyping is only a confidence-building approach unable to replace exhaustive analysis methods.

(2) All **static analysis** techniques have in common that they avoid the enumeration of the state space of a system. To get an overview on static analysis techniques of the "classical" Petri net theory as well as definitions of the terms in the following used, but not explained (for obvious reasons) see (Murata 1989) and (Starke 1990). Basic techniques corresponding mainly to general analysis (of boundedness or liveness) are net reduction and structural analysis.

Net reduction tries to decrease the net size by property-preserving replacing of local net sub-structures by smaller ones. A set of reduction rules is called complete if it reduces any net to a minimal prototype having the same properties (concerning boundedness and liveness), and it is called nearly complete, if it reduces at least all bounded and live (shortly well-formed) nets to its 2-node prototype. Nearly complete sets of reduction rules are known only for a quite restrictive net class - the extended free choice nets (Desel and Esparza 1995). For general Petri nets, we only have some quite weak reduction rules which usually do not help substantially.

Structural analysis consists in the analysis of at best locally structural properties, which allow - usually in a certain combination - conclusions on behavioural properties. A famous and maybe the most successful example is the so-called deadlock trap property, which involves liveness in case of extended simple nets, and deadlock freedom else.

Additionally, there exist different methods of **integer programming** based on a linear-algebraic description (incidence matrix) of the Petri net. Integer programming revealing net invariants supports general analysis (any well-formed net is covered by semipositive place and transition invariants) as well as special analysis. In the latter case, system invariants are proven by showing the existence of related net invariants. So first, suitable system invariants have to be hypothesized, and second, the related net invariants have to be found from the (in general non-minimal) basis of invariants provided by a net analysis tool. Generally, this is hardly manageable for larger systems (larger concerning the number of places).

Moreover, place invariants may be helpful to proof the unreachability of states by using the *token conservation equation*. A further sufficient condition for unreachability, which is similar efficiently to check, is given by the *state equation* (both equations are used in section 6.2).

Recently, new approaches appeared combining known concepts. The state equation test has been improved in (Melzer and Esparza 1996) by adding the test of the trap property. A marking has the trap property if it marks every trap that is marked at the initial marking. Obviously, each reachable marking satisfies not only the state equation, but also the trap property. Therefore, this combination yields a stronger condition for unreachability that can still be efficiently checked. In (Esparza and Melzer 1998) a detailed discussion of related implementation considerations can be found.

In (Lautenbach and Ridder 1995), structural knowledge of the given net (place invariants and traps) is used to accelerate model checking algorithms.

(3) If a desired system property can not be determined by static analysis techniques, **dynamic analyses** have to be tried. The classical approach is the exhaustive construction and exploration of the (interleaving) state space (reachability graph). The exploration may be controlled either by on-the-board questions to general net properties like boundedness, freedom of deadlocks, and liveness, respectively, or by a more sophisticated flexible query language. Such a query language encloses usually some version of temporal logics. For an introduction to temporal logics and the notation used in this paper see e.g. (Ben-Ari et al. 1983), (Emerson 1990).

Although almost every behavioural property of a Petri net with finite reachability graph can be theoretically decided by exhaustive analysis, this approach is limited in practice due to the state explosion problem. Basically, there are three techniques alleviating the state explosion dilemma.

(3.1) Compression techniques attack the state explosion by avoiding an explicit representation of the total (interleaving) state space of a concurrent system. There are two different approaches to represent state spaces in a memory efficient manner.

Modular representations of reachability graphs using Kronecker algebra have been successfully applied for performance analysis of stochastic Petri nets (Buchholz 1991), (Kemper 1997). In (Kemper and Lübeck 1998), this approach has been adopted to Computation Tree Logic (CTL) model checking (Clarke et al. 1986) of the state space of ordinary Petri nets. First computational results show that a *Kronecker representation* of the state space may be very space efficient if a suitable modularization can be found.

Ordered binary (or natural) *decision diagrams* (OBDDs, ONDDs) (Bryant 1986, 1992), (Lautenbach and Ridder 1995) represent sets of states (and sets of transitions between states) by their characteristic function. Model checking (in this context sometimes called *symbolic model checking*) of temporal logic formulae can be performed on OBDDs (ONDDs) without an

2 PETRI NET FRAMEWORK

Net based software engineering has been a well-know approach for more than 15 years. Basically, there are two different possibilities of the role Petri nets are able to play during the software development process.

When used right from the beginning, Petri nets are constructed *a priori* to model and prototype the concurrent aspects of the system under development, and the system designer is able to predict, at the chosen abstraction level, the possible (qualitative and quantitative) behaviour of the system under development. After being satisfied with the analysis result obtained, the actual program code (of the communication/ synchronization skeleton) in the usually given implementation language can be generated, or the sequential program parts are added directly to the Petri net and their execution is driven by the token flow.

The second approach to the use of Petri nets in concurrent software development relies on the *a posteriori* usually automatic Petri net generation from an high-level language description of the software under development (interpreted as specification, implementation, or anything in between). By analysing the generated formal model in the background, conclusions on the given software's properties are possible. For more details see e.g. (Balbo 1992), (Heiner 1992).

Independent of its place within the software development cycle, Petri net based software validation tries to minimize the presence of faults in the system's operation phase by analytical and (as far as possible) computer-aided methods in the pre-operation phase.

Concerning the type of properties to be validated, two classes of qualitative validation techniques can be distinguished:

- **Context checking** deals with general qualitative properties like freedom from data or control flow anomalies which must be valid in any system independent of its special semantics (for that reason, it is called in the following *general analysis*). These properties are generally accepted or project-oriented consistency conditions of the static semantics of any program structure like boundedness and liveness.
- **Verification** aims at special qualitative properties like functionality, safety or robustness, which are determined by the intended special semantics of the system under development (to underline this fact, it is called in the following *special analysis*).

Both general and special analysis aim at time-less properties which should be valid independent of time restrictions. Unfortunately, that is not always obvious in the case of concurrent systems. For a discussion of related problems see (Heiner and Popova 1997d).

Evidently, a successful general analysis is a necessary prerequisite to prove that some special qualitative properties will be fulfilled under any circumstances. So, the validation of qualitative properties can be divided into two consecutive steps, which supplement each other. First, the context checking of general semantic properties (general analysis) has to be done by a suitable combination of static and dynamic analysis techniques of Petri net theory. Afterwards, the verification of well-defined special semantic properties (special analysis) given by a separate specification of the functional and safety requirements has to be performed. Especially for the second step it is very useful to supplement the power of "classical" Petri net theory by the model checking approach, using temporal logic as a flexible query language for asking questions about the (complete/ reduced) set of reachable states.

The process model applied in our case studies can be seen as an adaptation of the general Petri net based approach to software validation presented in (Heiner et al. 1994). Key ideas are (compare figure 1):

- separate specifications of functional, safety and performance requirements which have to be provided by the customer of the system to be developed,
- a recommended order, in which validation methods should be applied (referring to figure 1 from top to bottom), and

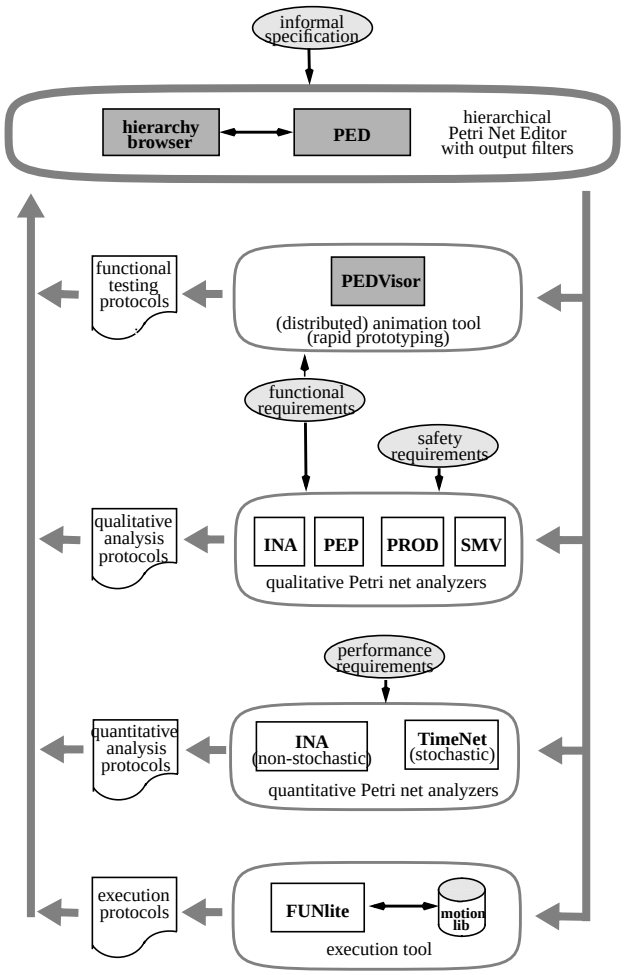

Figure 1: Tool box overview.

mh@informatik.tu-cottbus.de

Proc. High Perfomance Computing '98, Boston, April 1998,
session "Petri net Applications and HPC", pp. 394-403

PETRI NET BASED SYSTEM ANALYSIS WITHOUT STATE EXPLOSION *)

Monika Heiner
Brandenburg University of Technology at Cottbus,
Department of Computer Science, Postbox 101344, D-03013 Cottbus

mh@informatik.tu-cottbus.de
http://www.informtik.tu-cottbus.de

keywords: hierarchical place/transition nets, temporal logics, verification, static analysis, model checking, reactive systems;

ABSTRACT:

The development of provably error-free concurrent systems is still a challenge of practical system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. Among those Petri net analysis techniques suitable for strong verification purposes there is an increasing amount of promising methods avoiding the construction of the complete interleaving state space, and by this way the well-known state explosion problem.
These alternative approaches are summarized and compared with each other: structural analysis, integer programming, compressed and composite state space representations, lazy state space constructions, and partial order representations.
It is demonstrated by means of case studies that the available methods and tools are actually applicable successfully to at least medium-sized systems. For that purpose, the step-wise validation of various qualitative system properties (consistency, safety, progress) of the concurrent control software of reactive systems is exemplified. If possible, different analysis techniques are applied and compared with each other concerning their pros and cons. The main lesson learnt is that the different methods do not compete, but complement each other.
Finally, objectives of an open integrated tool box to support Petri net based dependability engineering are outlined.

1 INTRODUCTION

Petri nets enjoy several advantages with respect to modelling and analysis of discrete event systems with inherent concurrency. Worth mentioning is especially the ability of combining different methods on a common representation. This variety ranges from informal (animation) via semi-formal (systematic testing) up to formal (exhaustive analysis) methods and comprises qualitative as well as quantitative evaluation techniques. But maybe most valuable is the fact that among the formal methods suitable for strong verification purposes there is an increasing amount of promising methods avoiding the construction of the complete interleaving state space, and by this way the well-known state explosion problem.

*) This work is supported by the German Research Council under grant ME 1557/1-1

This paper gives an overview on these alternative methods and reports on our experience concerning their strength and limitations for verification of medium-sized reactive systems.

The discussion covers

- **static analysis techniques**, constructing no state space at all,
- **compression techniques**, representing the state spaces in a memory efficient manner,
- **lazy state space constructions** building reduced state spaces instead of the complete ones, and
- **alternative state space constructions**, exploiting concurrency to build partial order (true concurrency) descriptions of the system behaviour.

It is claimed that the optimistic analyses results gained up to now are typical for at least a certain class of practical problems. This assumption is justified by case studies performed, in which Petri net specifications of realistically sized controller software have been developed and verified.

This paper deals mainly with implementable *qualitative* analysis techniques suitable for place/transition nets without time constraints. The reader not familiar with this model see for an introduction e.g. (Peterson 1981) or (Reisig 1985). *Quantitative* evaluations of our case studies are still under progress. We are experimenting with different types of time-dependent Petri nets, e.g. duration interval nets to prove the meeting of given deadlines in the framework of worst-case evaluation (Heiner and Popova 1997b), and stochastic nets to estimate probability measures like throughput, bottlenecks or average processing time (Wikarski and Heiner 1995).

The paper is organized as follows. Section 2 gives an overview on the underlaying Petri net based framework to develop dependable software based systems. Section 3 summarizes Petri net related analysis techniques and corresponding analysis tools, which are part of our current tool box. Typical examples of performed case studies and their informal requirement specifications are given in section 4. Afterwards, the applied way of modelling with hierarchical Petri nets is outlined in section 5. The step-wise validation of qualitative properties is described in more detail in section 6. Finally, some conclusions into the direction of an open integrated tool box to support Petri net based dependability engineering are summarized in section 7.