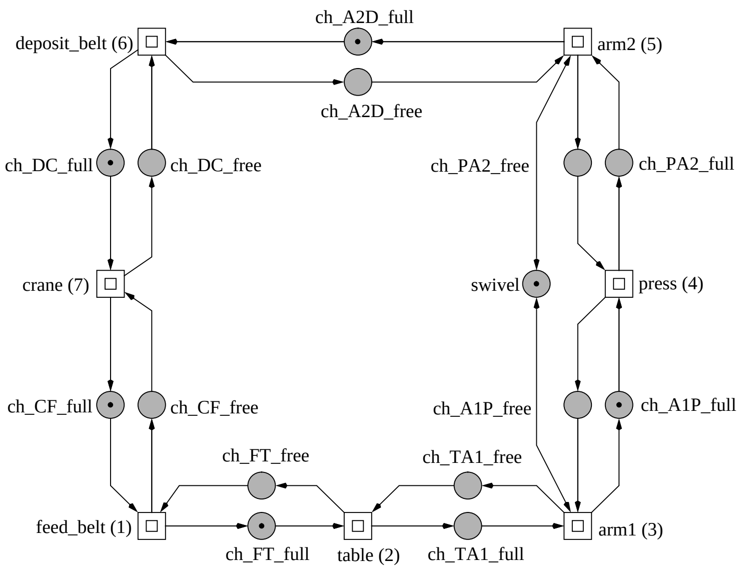
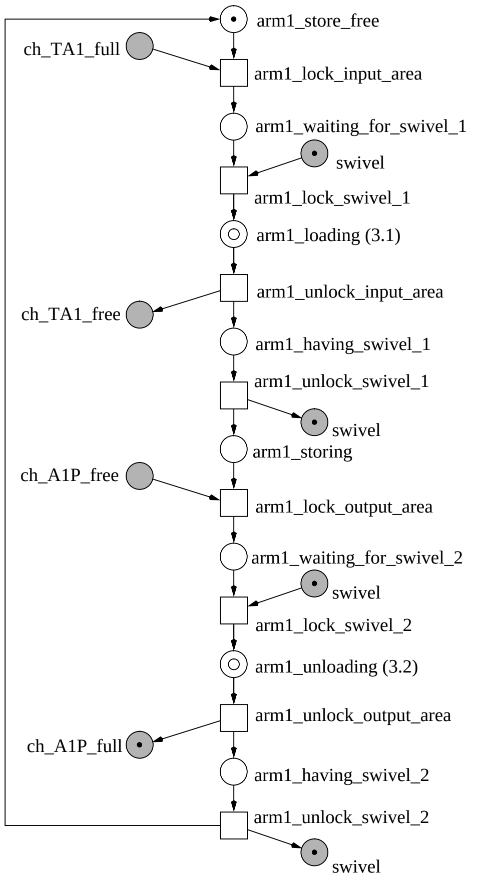
Appendix: Petri net of the production cell, control model - overview.

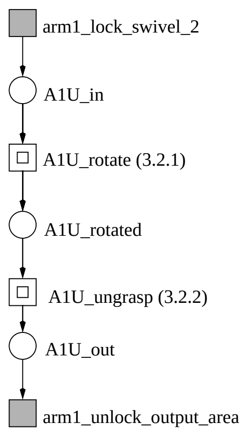
(0) top level:



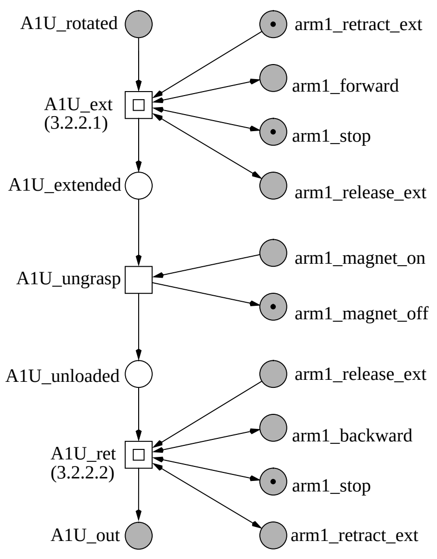
(3) arm1:



(3.2) arm1_unloading:



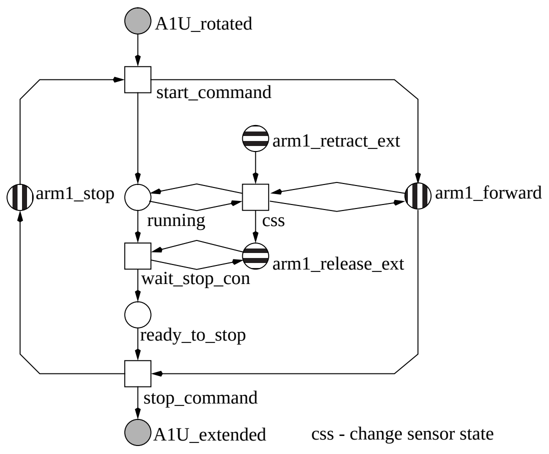
(3.2.2) A1U_ungrasp:



Statistics:

- The total hierarchically structured Petri net consists of 232 places and 202 transitions, devided into 65 nodes of the hierarchy tree.
- There are 37 macro transitions, containing the elementary motion steps (e.g. 3.2.2.1) and forming the sheets of the hierarchy tree.

(3.2.2.1) A1U_ext:



css - change sensor state

Assumptions on the environment model (left):

- A start command will force the associated device to change from an inactive (arm1_stop) to an active (arm1_forward) state (a stop command vice versa).
- The performed motion will cause eventually the change from the initial sensor value (arm1_retract_ext) to a final sensor value (arm1_release_ext), indicating that the motion has been completed.

Drawing convention (right):

Shaded nodes are so-called logical nodes. They serve as connectors to avoid immoderate edge crossing. All logical nodes with the same name are logically identical.

macro nodes:

◎ macro place

⊡ macro transition

logical nodes:

■ ●

special logical nodes:

◍ actuator states

⊜ sensor states

For the cooperation model, this requirement is expressed by the CTL formula

$$\mathsf{AG}\,(\mathrm{arm1\_loading} \vee \mathrm{arm1\_unloading} \rightarrow \neg\mathrm{press\_go\_loadpos} \vee \neg\mathrm{press\_go\_unloadpos})$$

Because the cooperation model does not comprise an environment model, system properties are only expressible in terms of internal system states (names of places holding a token) like “arm1_loading” or “press_go_loadpos”.

If a model of the behaviour of the system environment is included in the system description, properties can be expressed in terms of this environment model. Within the control model, the requirement mentioned above is expressed by the LTL formula

$$\mathsf{G}\,(\mathrm{arm1\_release\_angle} \wedge \mathrm{arm1\_release\_ext} \rightarrow \mathrm{press\_stop} \wedge \neg\mathrm{press\_at\_upper\_pos})$$

The major difference between the two formulae is that the latter contains no references to internal system states. In the same way, most of the safety requirements for the production cell can be expressed on the base of a few simple and - by means of a net model - well-documented assumptions on the environment behaviour.

6 Final Remarks

Up to now, a Petri net model to control the given production cell has been developed which enjoys provably a lot of valuable qualitative properties - general as well as special ones. Beyond that, the following investigations are in preparation:

- quantitative analysis by different types of time-dependent Petri Nets:
  by Duration Interval Nets [17] based on Interval Nets [19] to prove the meeting of given deadlines by worst-case evaluation, and by Stochastic Nets [27] to estimate throughput or average processing time;
- synthesis of the actual control software based on [20].

Throughout our case study, (the rarely available and rather restrictive) compositional approaches of Petri net analysis have not been discussed yet. They have been skipped in order to get a feeling for the borders of those net/state space sizes, which are actually manageable by available analysis tools.

References

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

[2] BEST, E., GRAHLMANN, B., PEP - Programming Environment Based on Petri Nets, Documentation and User Guide: Univ. Hildesheim, Institut für Informatik, Nov. 1995.

[3] CASAIS, E., Eiffel; A Reusable Framework for Production Cells Developed with an Object-oriented Programming Language, in: Lewerentz, C., Lindner, T., ed.: *Case Study “Production Cell” A Comparative Study in Formal Software Development*, FZI-Publication 1/94, Forschungszentrum Informatik, Karlsruhe 1994, 241-256.

[4] CLARKE, E. M., EMERSON, E. A., SISTLA, A. P., Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Trans. on Programming Languages and Systems **8**(1986)2, 244-263.

[5] CZICHY, G., Design and Implementation of a Graphical Editor for Hierarchical Petri Net Models (in German), Diploma Thesis, TU Dresden - GMD/FIRST, 6/1993.

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

[7] EMERSON, E. A., Temporal and Modal Logic, in: J. v. Leeuwen, ed.: *Handbook of Theoretical Computer Science*, Vol. B, Elsivier, Amsterdam 1990, 995-1072.

[8] ENGELFRIET, J., Branching Processes of Petri Nets, Acta Inf. 25(1991), 575-591.

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

[10] ESPARZA, J., Model Checking Using Net Unfoldings, Science of Computer Programming **23**(1994), 151-195.

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

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

[13] HEINER, M., VENTRE, G., WIKARSKI, D., A Petri Net Based Methodology to Integrate Qualitative and Quantitative Analysis, Information and Software Technology **36**(94)7, 435-441.

[14] HEINER, M., WIKARSKI, D., An Approach to Petri Net Based Integration of Qualitative and Quantitative Analysis of Parallel Systems, Techn. Report BTU Cottbus, I-09/1994, Dec. 1994.

[15] HEINER, M., Petri Net Based Software Dependability Engineering, Tutorial Notes, Int. Symposium on Software Reliability Engineering, Toulouse, Oct. 1995.

[16] HEINER, M., DEUSSEN, P., Petri Net Based Qualitative Analysis - a Case Study, Techn. Report BTU Cottbus, I-08/1995, Dec. 1995.

[17] Heiner, M., Popova, L., Worst-Case Analysis of Concurrent Systems with Duration Interval Petri Nets, Techn. Report BTU Cottbus, I-02/1996, February 1996.

[18] LEWERENTZ, C., LINDNER, T., *Formal Development of Reactive Systems - Case Study Production Cell*, LNCS 891, 1995.

[19] POPOVA, L., On Time Petri Nets, J. Information Processing and Cybernetics EIK **27**(1991)4, 227-244.

[20] SCHWIDDER, K., Petri Net Based Modelling and Simulation of Automation Techniques’ Discrete Processes (in German); in Scheschonk, G.; Reisig, W. (eds.): *Petri Net Applications for Design and Development of Information Systems*, Springer 1993, pp. 209-221.

[21] STARKE, P. H., *Analysis of Petri Net Models* (in German), Teubner, Stuttgart 1990.

[22] STARKE, P. H., INA - Integrated Net Analyzer, Manual (in German), Berlin 1992.

[23] VALMARI, A., A Stubborn Attack on State Explosion, Formal Methods in System Design **1**, (1992) 4, 297-322.

[24] VALMARI, A., Alleviating State Explosion during Verification of Behavioral Equivalence, Univ. of Helsinki, Department of Computer Science, Report A-1992-4, Helsinki 1992.

[25] VARPAANIEMI, K., On Computing Symmetries and Stubborn Sets, Helsinki Univ. of Technology, Digital Systems Laboratory Report B 12, Espoo 1994.

[26] VARPAANIEMI, K., HALME, J., HIEKKANEN, K., PYSSYSALO, T., PROD Reference Manual, Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, August 1995.

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

Table 1: Size of analyzed nets and analysis efforts done (cooperation model).

	places/ transitions	Analysis (INA)		Execution Times			
		DTP[a)]	R_{stub}[b)]	R[c)]	INA[d)]	INA[e)]	PROD[f)]
table / press							
with init part	13 / 9	(N)	12	28	1.5’	1.54’	7.96’
without init part	12 / 8	28	8	24	1.46’	2.16’	7.41’
crane	12 / 8	31	11	48	1.62’	2.12’	7.39’
arms							
version 1	13 / 8	38	11	48	1.5’	3.12’	14.78’
version 2	17 / 12	109	15	112	1.73’	1.65’	24.92’
version 3	17 / 12	88	15	96	1.77’	1.88’	24.19’
belts	12 / 8	26	8	36	1.58’	2.23’	7.38’
subsystem with							
arm version 1	25 / 16	175	47	640	7.81’	3.64’	14.78’
arm version 2[g)]	33 / 24	3.851 (N)	75	1.984	53.23’	7.78’	24.92’
arm version 3	33 / 24	725	140	1.800	57.07’	8.12’	24.19’
open system	51 / 36	1145	299	77.760	86132.6’	3653.6’	1073.35’
closed system	51 / 36	1140					
with 1 plate			36	864	18.58’	6.07’	25.93’
with 2 plates			72	4.776	365.53’	18.77’	56.21’
with 3 plates			94	12.102	2401.28’	87.1’	125.71’
with 4 plates			98	16.362	4167.93’	150.85’	169.15’
with 5 plates			121	12.144	2373.1’	78.6’	123.8’

a) processed candidates to check the deadlock trap property
b) number of states of the stubborn reduced reachability graph $\mathbf{R}_{\mathbf{stub}}$
c) number of states of the complete reachability graph **R**
d) time effort to generate **R** (with coverability test)
e) time effort to generate **R** (without coverability test)
f) time effort to generate **R**
g) contains dead states

reachability graph analysis, the model checking approach of PEP is based on the construction of a so-called *finite prefix of a branching process* [10], which is only suitable for 1-bounded Petri nets. However, both the finite prefix construction and the model checking work extraordinary fast: The liveness of transitions can be checked within the insignificant time amount of 0.3 seconds! Similar results are obtained for those safety properties (about the half of the required) which are expressible in this logic.

Several required safety properties of the control model have been proven successfully using PROD’s stubborn set based on-the-fly verification method. The following conclusions are worth mentioning:

1. The existence of an environment model allows to express both assumptions on the behaviour of physical machines and system properties in terms of the influence of a system on its environment. Evidently, requirements expressed in this way are comprehensible without a deeper knowledge of the internal structure of a system.
2. Because of the structure of the environment model, it is impossible to express properties related to illegal states of single devices. A more improved model must contain explicit error states.
3. The stubborn set based on-the-fly verification method has been proven to be applicable even for systems with a larger state space. The analysis efforts were between 2 and 25 minutes for the formulae for which a model checking were done. However, certain properties like the liveness of transitions cannot be expressed in LTL because of the lack of a quantification on computation paths.
4. Although the expressive power of PEP’s temporal logic is rather restrictive, simple liveness and safety requirements can be verified very efficiently. Therefore, the model checking techniques provided by PEP and PROD appear to be complementary to each other.

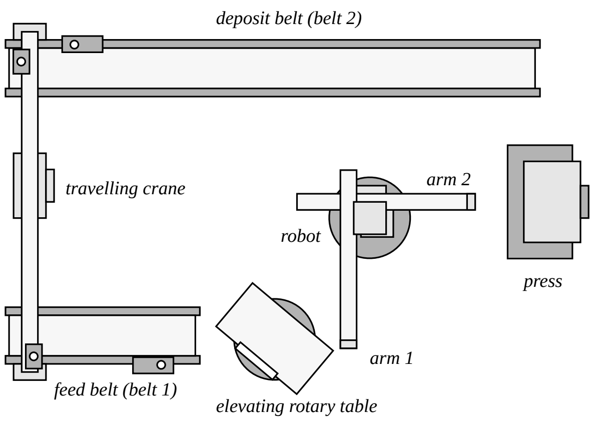
5 Examples of Verified Safety Properties

We will illustrate the different levels of reasoning on system behaviour of the cooperation and control models. For the following, see the appendix which contains an overview on the structure of the Petri net model of the production cell.

One safety property required for the controller software of the production cell is:

> *The press may only close when no robot arm is positioned inside it.*

Figure 1: Top view of the production cell



abstraction levels. The more abstract **cooperation model** describes the synchronization of the machine controllers. This model has been influenced essentially by [3]. The construction of the model was done step-wise. At first, common patterns concerning the intended communication behaviour of the controllers for the physical devices have been identified and modelled as Petri nets. Then, the complete cooperation model was constructed by composition of instances of these communication patterns by merging certain communication places (see Appendix, the (0) top level net).

General analysis was done using INA, while for special analysis certain requirements were expressed in CTL and proven for the complete state space of the composed model using PROD.

After designing and analyzing successfully the cooperation model, a refinement has been done which incorporates the interactions of the controllers with the interface (actuators, sensors) of the production cell. Furthermore, this **control model** comprises a Petri net description of the environment which allows to express certain assumptions on the behaviour of the physical devices in response to the controllers' actions. As before, the construction of the model was done bottom-up: A net structure for an elementary control procedure was defined which involves the controller part as well as the environment part of one basic motion step of one device. One basic step consists of (compare 3.2.2.1 in the Appendix):

- the activation of some device (transition *start_command*),
- the reception of a certain sensor value that indicates that the motion is completed (transition *wait_stop_con*), and
- the deactivation of the device (transition *stop_command*).

More complex motion step controls were constructed by composition of elementary ones (see 3.2.2 and 3.2 in the Appendix).

Again, INA and PROD were applied for general and special analysis. Because of the extraordinary size of the state space of the control model, the time and space effort to generate the complete reachability graph became unacceptable. For this reason, safety requirements were expressed in LTL. While PROD supports model checking for CTL only by the complete reachability graph of a Petri net, LTL formulae not containing the nexttime operator can be checked very efficiently by construction of a reduced state space which is in so-called *CFFD-equivalence* to the complete state space using the stubborn set method (see [6, 11, 23, 24, 25]). PROD provides a so-called on-the-fly verification method based on this approach.

4 Summary of the Main Analysis Results

Cooperation model. In general analysis, we are basically interested in proving the general properties boundedness and liveness. Using INA, both properties can be decided efficiently by showing that the net under consideration are covered by semipositive place invariants and by proving the deadlock trap property, respectively, because of certain structural properties of the corresponding net models (marked graphs or extended simple nets). Dead states in one discussed controller version have been found very fast by the stubborn set reduction method. Table 1 compiles some details on the size of the analyzed nets and the analysis efforts done. Please note especially the impressive reduction effect inherent to the stubborn set method. The analysis was done on a SUN sparc station 5.

The reachability graph analyzer PROD has been successfully applied to prove certain safety properties of the cooperation model given by CTL formulae. Two major problems have been arisen:

1. The expressibility of certain properties. Due to the lack of appropriate terms of the behaviour of physical devices, requirements which are given explicitly in terms of the system's interaction with the environment (like sensor values) cannot be expressed only in terms of internal system states.
2. The time effort for model checking. Safety properties of the general logical form $\mathsf{AG}\,(\neg\varphi)$ (φ a state formula) can be checked within an acceptable amount of time (about a few minutes). For systems with a medium-sized state space, the time effort to validate more complex formulae like progress properties of the form $\mathsf{AG}\,(\varphi \rightarrow \mathsf{AF}\chi)$ (φ, χ state formulae) is extraordinary large (about ten hours on a SUN sparc station 20).

Control model. For the control model, a complete construction of it's state space became impossible (we have stopped the generation after about 2 days, having generated more than 700.000 states). In spite of this, boundedness is still decidable very efficiently by showing that the net model is covered by semipositive place invariants (again it took only a few seconds). Due to the added environment behaviour, all discussed net models exhibit a net structure beyond the extended simple one. So, the deadlock trap property could only show the freedom of dead states. However, the construction of the stubborn set reduced reachability graph is still manageable even for larger systems with unknown size of the complete state space. So, the prove of the deadlock freedom seems to be practicable in any case.

It was impossible to prove liveness using INA or PROD, because of the lack of suitable structural properties and the size of the state space. The liveness of a transition *t* can be expressed in CTL by the formula $\mathsf{AG}\,\mathsf{EF}\,(en\,(t))$ ($en\,(t)$ denotes the proposition that all preconditions of a transition *t* are satisfied), but not analyzed by PROD. In LTL, only a stronger property related to the livelock freedom of transitions can be expressed by $\mathsf{G}\;\mathsf{F}\,en\,(t)$, and actually analyzed by PROD's on-the-fly verification. Unfortunately, not (but almost) all transitions in the control model enjoy this property.

Recently we have gained some results with one of the model checking components of PEP, which supports a fragment of CTL containing only the temporal operators AG and EF. Hence, it is impossible to express progress properties in this logic. Instead of 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 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. Unfortunately, that is not always obvious in the case of concurrent systems.

Evidently, a successful general analysis is a 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, e. g.

- animation by playing the token game,
- static analyses by net reduction, structural analysis or net invariant analysis,
- dynamic analyses by complete/reduced construction of the system's state space (reachability graph),
- model checking of temporal formulae.

Afterwards, the verification of well-defined special semantic properties (special analysis) given by a separate specification of the required functionality 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.

Net-based **animation** aims at functional behaviour simulation by playing the token game. 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.

All **static analysis** techniques have in common that they avoid the construction of the reachability graph. Reduction and structural analysis (e.g. the deadlock trap property) correspond mainly to general analysis (of (un-) boundedness or (un-) liveness). The invariant analysis supports general analysis (if the net is covered with semipositive place invariants) as well as special analysis. In the latter case, program invariants are proven by showing the existence of related net invariants. So first, suitable program 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 size of states).

Dynamic analysis techniques have to be used if the static analysis efforts were not successful, or if properties are wanted which cannot be analyzed statically at all (e.g. reversibility, livelock freedom, dynamic conflicts - especially those conflicts, where communication places are involved, firing of facts, etc.). Due to the well-known effect of state explosion, lazy construction methods to consider only reduced state spaces have been elaborated. E.g. the stubborn set reduced reachability graph is usually (in the case of highly concurrent systems) much smaller than the complete one, but exhibits all dead states, if any.

For verification purposes, **temporal logic** is widely accepted as a flexible language for the specification of required system behaviour. In *Linear Time Temporal Logic* (LTL) [7], the classical (propositonal) logic is augmented with temporal operators, for instance X, F, and G. The course of time is regarded to be linear, i. e. for a given system state, LTL formulae express properties for every possible future system behaviour. $\mathsf{X}\varphi$ states that φ is true at every immediate successor state, $\mathsf{F}\varphi$ means that φ will eventually be true, while $\mathsf{G}\varphi$ expresses that henceforth φ holds.

In *Computational Tree Logic* (CTL) [1], time is regarded to be tree-like structured: The temporal operators are combined with the *path quantifiers* A (formula holds for every future behaviour) and E (formula holds for at least one future behaviour).

In **model checking**, the reachability graph (in different versions) of a Petri net is interpreted as a model of the behaviour of the net, and the validity of a temporal formula in this model is checked by inspection. All properties can be checked which are expressible in those versions of temporal logics which are provided by the tools used. In this way, even very large reachability graphs become manageable.

The **tool kit** used is as follows: The graphical Petri net editor PED is based on [5] and is still under development to be adapted to current wishes. It supports basically the construction of hierarchical place/transition nets. Complementary, all necessary attributes of those net types can be assigned, which are analyzable by INA. The analysis tools in use INA [22], PROD [26] and PEP [2] have to be understood as implementations of well-proven theorems of Petri net theory. Not included in [16] are our experience of the qualitative analysis power of PEP, which has become available lately. At the end of section 4 we give a brief discussion of the results gained in the meantime.

3 Case Study

The focus of our investigations builds a (really existing) industrial facility. This production cell 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 1). Their common goal is the transport and transformation of metal plates. For details the reader is referred to [18]. In that paper, 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 analyzed the control software in two

1) The investigations presented in [16] are actually a mixture of both approaches. We started with the second one by following the Eiffel solution presented in [3]. Encouraged by the clear net structures and the analysis results gained, we were optimistic that also a complete Petri net-based development of the control software should be possible. So we changed to the approach mentioned firstly.

A Case Study in Design and Validation of Reactive Systems by Means of Petri Nets

Monika Heiner, Peter Deussen

Brandenburg Technical University of Cottbus
Department of Computer Science
Postbox 101344
D-03013 Cottbus
Germany
mh@informatik.tu-cottbus.de, pd@informatik.tu-cottbus.de
Phone: (+ 49 - 355) 69 - 3885
Fax: (+ 49 - 355) 69 - 3830

Abstract. The development of provably error-free concurrent systems is still a challenge of system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. To evaluate the reached practicability degree of available methods and tools to at least medium-sized systems, we demonstrate the step-wise development and validation of the control software of a reactive system. The validation of qualitative properties comprises two steps. At first, context checking of general semantic properties is done by a suitable combination of static and dynamic analysis techniques of Petri net theory. Afterwards, verification of well-defined special semantic properties, especially safety properties, given by a separate specification of the required functionality, is performed by model checking. Strong emphasis has been laid on automation of the analyses to be done.
This paper provides a brief outline of the authors' work which is described in detail in [16].

Keywords: Parallel software/system engineering, static analysis, formal methods, Petri nets, temporal logics, control software, reactive system, discrete event systems, reliability.

1 Introduction

The development of provably error-free concurrent systems is still a challenge of system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. To evaluate the reached practicability degree of available methods and tools to at least medium-sized systems, the authors demonstrate in [16] the step-wise development and validation of the control software of a reactive system - a (really existing) production cell in a metal-processing plant [18]. In this paper, a brief outline of our work is presented.

The validation of qualitative properties comprises two steps. At first, context checking of general semantic properties is done by a suitable combination of static and dynamic analysis techniques, mainly of "classical" of Petri net theory. Afterwards, verification of well-defined special semantic properties, especially safety properties, given by a separate specification of the required functionality is performed. Strong emphasis has been laid on automation of the analyses to be done.

The case study has been designed to evaluate different kinds of software development methods. The objective is to develop verified control software, taking into account various safety conditions and performance constraints. This case study has been already investigated by a lot of different formal methods [18], but up to now Petri nets have been used only as supplementing description. This paper provides the possibility to consider also a strongly Petri net-oriented approach to compare advantages and drawbacks inherent to different formal methods.

This paper is organized as follows. Section 2 gives a short overview of the Petri net based framework of software validation and introduces some basic terminology. The essential points of the discussed case study concerning the task description and the modelling of the controller components for the production cell are presented in section 3. Section 4 gives a short overview of the main analysis results obtained, while section 5 shows two typical examples of the verified safety properties. Finally, section 6 comprises an outlook on further work in preparation and open questions, which have to be solved in order to improve the Petri net based validation.

2 Petri Net Based Framework of Software Validation

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 developer is able to predict, at the chosen abstraction level, the possible behaviour of the system. After being satisfied with the analysis result obtained, the program code (of the communication skeleton) in the usually given implementation language can be generated.

The second approach to the use of Petri nets in concurrent software development relies on the a posteriori generation of Petri nets from an high-level language description of the software under development (interpreted as specification, implementation, or anything in between), see e. g. [15].

Independent of its place within the software development cycle$^{1)}$, validation tries to minimize the presence of faults in the operation phase by analytical and (as far as possible) computer-