Proc. 1st Int. Workshop on Manufacturing and Petri Nets, Osaka, June ’96, pp. 177-196,
held at Int. Conf. on Application and Theory of Petri Nets (ICATPN ‘96).

A Case Study in Developing Control Software of Manufacturing Systems with Hierarchical Petri Nets

Monika Heiner, Peter Deussen, Jochen Spranger
Brandenburg Technical University of Cottbus
Department of Computer Science
Postbox 101344
D - 03013 Cottbus
Germany
{ mh, pd, jsp }@informatik.tu-cottbus.de
Phone: (+ 49 - 355) 69 - 3885
Fax: (+ 49 - 355) 69 - 3830

Abstract. The application of Petri nets is one of the well-known approaches to develop provably error-free control software of manufacturing systems. To evaluate the reached practicability degree of available methods and tools to at least medium-sized systems, a case study has been performed to develop modularized control software of a production cell with hierarchical Petri nets, supporting reuse as well as step-wise validation.

Keywords: Concurrent system engineering, hierarchical Petri nets, reusable components, process model, validation, static analysis, temporal logics, performance evaluation, simulation, control software, manufacturing software.

1 Introduction

The development of provably error-free software-based concurrent systems is still a challenge of system engineering. Design 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 step-wise Petri net-based development, comprising design and validation, of the control software of a reactive system - a (really existing) production cell in a metal-processing plant [20] - has been done.

The main objectives have been to develop modularized control software, supporting reuse as well as step-wise validation taking into account various safety conditions and performance constraints. For that purpose, the hardware/software interactions had to be modelled too.

The Petri net based validation comprising qualitative as well as quantitative properties

has been divided into several steps. First, the context checking (also called *general analysis*) of general semantic properties (like boundedness and liveness) was managed, basically by “classical” Petri net theory. Second, the verification of well-defined special semantic properties, progress as well as safety properties, given by a separate requirement specification was performed by model checking of temporal formulae (shortly called *special analysis*). Afterwards, quantitative analysis has been started by means of two different types of time-dependent Petri Nets - by Duration Interval Nets to prove the meeting of given deadlines (worst-case evaluation), and by Stochastic Nets to estimate typical performance measures (like throughput, processing time). These validation steps have to be applied repeatedly according the step-wise refinement of the system under development. Strong emphasis has been laid on automation of all the analyses to be done.

Finally, the actual control software has been generated automatically from the Petri net specification by using a library of auxiliary procedures necessary for elementary motion steps.

The purpose of this paper is to summarize the main results gained up to now and to highlight essential problems still to be solved.

The paper is organized as follows. Section 2 gives a short overview of the applied Petri net based process model to develop control software of manufacturing systems and the related tool kit currently in use. Section 3 describes the way of modelling with hierarchical Petri nets by composing a very small set of reusable Petri net components. The essential points of qualitative analysis, divided into general and special analysis, are summarized in section 4, while the synthesis of the actual control software is described in section 5. Finally, section 6 summarizes lessons learnt and conclusions in which direction to go ahead.

2 Petri Net Based Process Model

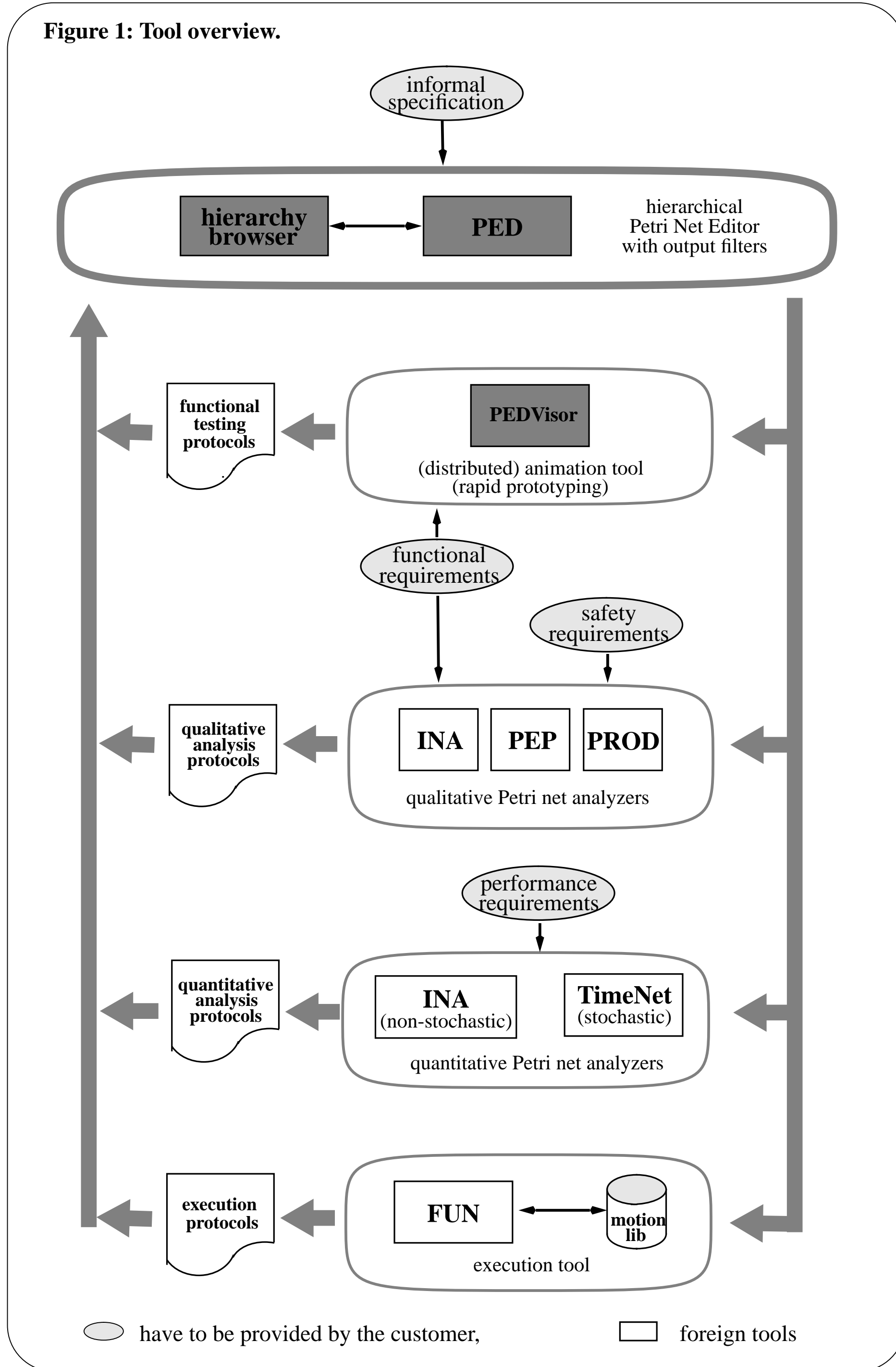
When used right from the beginning, Petri nets are constructed to model and prototype the concurrent aspects of the system under development, and the developer is able to predict, at the chosen abstraction level, the possible (qualitative and quantitative) behaviour of the system. After being satisfied with the analysis result obtained, the 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 implementation presented in this paper (see section 5) follows the second strategy.

The process model applied in this case study can be seen as an adaptation of the general Petri net based approach to software validation presented in [16], [17]. 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,

Proc. 1st Int. Workshop on Manufacturing and Petri Nets, Osaka, June '96, pp. 177-196,
held at Int. Conf. on Application and Theory of Petri Nets (ICATPN '96).

Figure 1: Tool overview.



- a recommended order, in which validation methods should be applied (referring to figure 1, from top to bottom), and
- the integration of qualitative as well as quantitative analysis on the basis of a common representation of the system under development.

The tool kit currently used is as follows. The Petri net EDitor PED with its hierarchy browser based on [8] 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 analysable by the evaluation tools linked up:

- PEDVisor [22][1)] allows to animate the functional behaviour by playing the token game.
- INA [27] 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.

  The next two 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 reachability graphs become manageable. But, the reachability graph has to be finite for that purpose. So, boundedness is here an unavoidable precondition.
- PEP [3] offers, besides many other interesting things not used here, a promising evaluation method (finite prefix of branching processes) for a certain type of temporal logic. Its application is however restricted to 1-bounded nets.
- PROD [32] supports *computational tree logic* (CTL, see e.g. [2]) as well as *linear time temporal logic* (LTL, see e. g. [10]). The evaluation of CTL formulae relies on the complete reachability graph. LTL formulae not containing the nexttime operator can be checked very efficiently by the construction of a reduced state space which is in so-called *CFFD-equivalence* [30] to the complete state space using the stubborn set method [30]. PROD provides an on-the-fly verification method based on this approach.
- TimeNet [14] supports the evaluation of generalized stochastic Petri nets by simulation as well as by analysis based on Markovian processes.
- FUN [25] allows the generation and (token-driven) processing of executable code.

The tool kit runs on UNIX with X11/Motif Interface (and on LINUX - with the exception of TimeNet)[2)].

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

1) still under development to be adapted to current wishes, e.g. visualization of analysis results;
2) All interested readers are invited to attend (on-line) demonstrations of the development steps outlined in this paper during the session break.

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 depends generally on the application area.

3 Modelling with Hierarchical Petri Nets

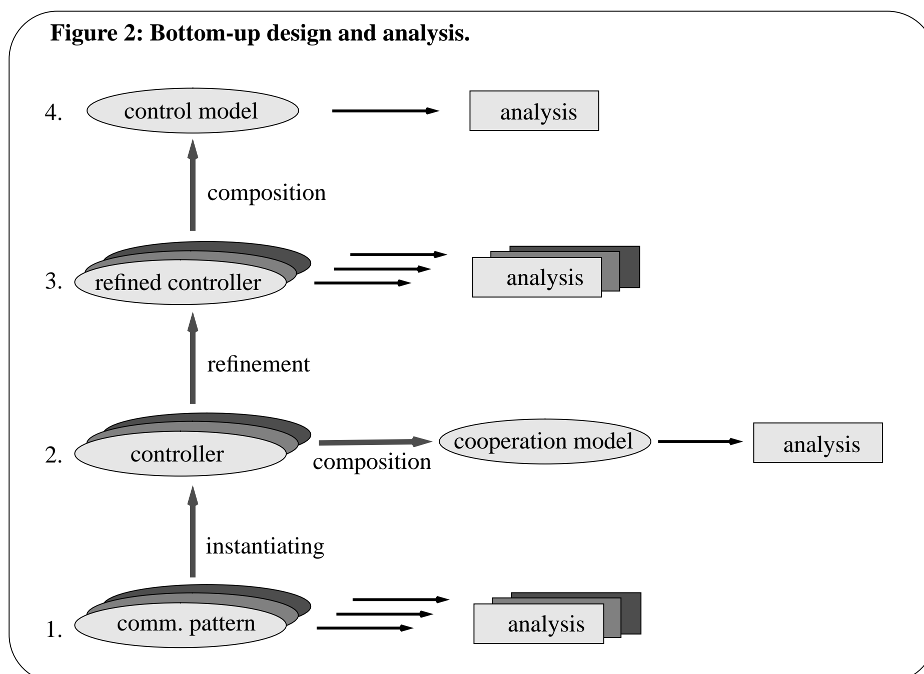
The focus of our investigation builds a (really existing) industrial facility. This production cell comprises six physical components: two conveyor belts, a rotatable robot equipped with two extendable arms, an elevating rotary table, a press, and a travelling crane. The machines are organized in a (closed) pipeline. Their common goal is to transport and transform metal plates. For details the reader is referred to [20], including also a list of safety and progress requirements which are to obey by any control program implementation.

3.1 General Procedure

We have developed and analysed the control software step-wise in two abstraction levels (compare figure 2) constituting the cooperation model (see section 3.2) and the control model (see section 3.3).

The more abstract *cooperation model* describes the synchronization of the machine

Figure 2: Bottom-up design and analysis.



controllers. The construction of the model was done bottom-up in the following way. At first, (three) general reusable patterns concerning the intended communication behaviour of the controllers for the physical devices have been identified and modelled as Petri nets (communicating state machines) inspired by [4]. These communication patterns have been analysed first. Then, the complete model was constructed by composition of instances of these communication patterns via merging so-called communication places.

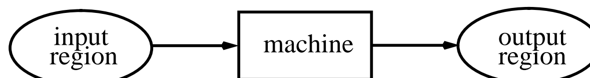
After having analysed successfully the cooperation model, refinements (of places as well as of transitions) have been done modelling the interactions of the controllers with the hardware interface (actuators, sensors) of the production cell. Furthermore, this *control model* comprises a Petri net description of the environment. As before, the construction of the model was done bottom-up: A general net structure for an elementary control procedure was identified which involves the controller part as well as the environment part of one basic motion step of any device type. More complex processing step controls were constructed by composition of elementary ones. After having modelled and analysed the refined controllers separately, the control model has been composed as already described above.

It is worth noting that the whole net has been constructed systematically using extensively a very small set (exactly seven) of reusable components. Therefore, similar control applications can be configured efficiently in a very short time period. The total net which can be found in [18] consists of 231 places and 202 transitions structured into 65 nodes of the hierarchy tree.

3.2 Cooperation Model

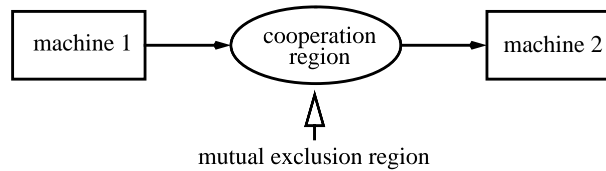
The manufacturing system considered consists logically of seven loosely coupled machine controllers acting to a high degree independently of each other. These machine controllers are organized in a (closed) pipeline to realize the transportation/transformation of metal plates. For that purpose, neighbouring machine controllers communicate with each other according a synchronous producer/consumer relation. There is neither a central controller of the production cell responsible for activating and deactivating the machines nor a global observer with full knowledge of the state of the production cell and of the metal plates.

Each machine follows a similar operation pattern: fetch a metal blank from the input region, process it, and deposit the plate on the output region. In order to do that, each machine performs cyclically a certain sequence of motions (of course synchronized according the states of its neighbours).



If two machines are connected, the output region of the predecessor and the input region of the successor merge to a cooperation region between two consecutive machines.

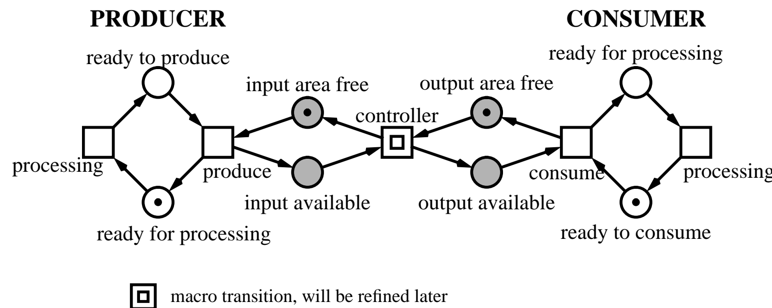
Obviously, such a cooperation region has to be organized as a mutual exclusion region, i. e. either the predecessor is allowed to put a plate into the region or the successor has the



access rights to take a plate from the region. But the concurrent access to the cooperation region by the adjacent machines is forbidden.

Furthermore, due to the given machine equipment, a cooperation region does not have any buffering capabilities. So the predecessor is allowed only to put a plate into the region, if the region is free, and the successor can only take a plate from the region, if it is full (see figure 3).

Figure 3: Producer consumer relation.



Additionally, there are two kinds of mutually exclusive shared resources.

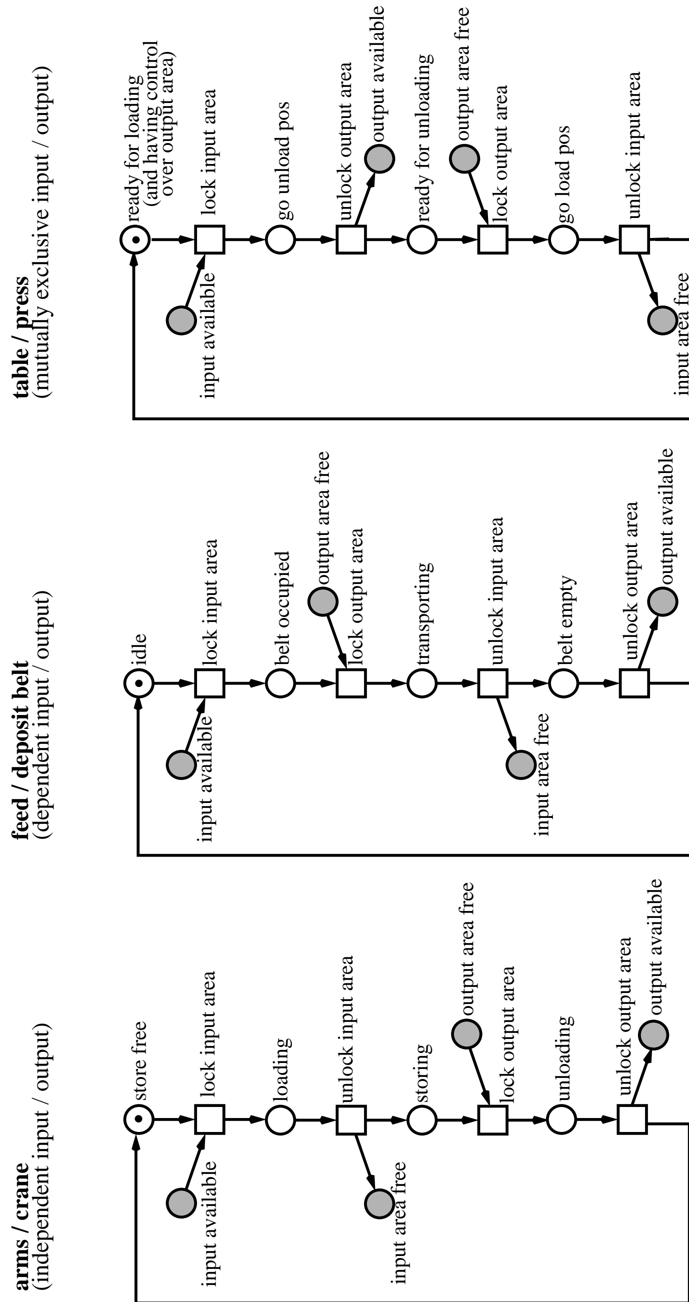
The robot arms are organized separately to enlarge the possible degree of parallelism within the cell (which may possibly result into an higher throughput). Doing so, the robot swivel (the engine to rotate both arms), becomes a shared resource of both arms, which can be used exclusively only.

There exist shared physical regions (intersection of trajectories of different machines). To avoid machine collisions, such shared physical regions have to be used exclusively. In our case study, the trouble disappears in a constructive way by the ad hoc requirements that the robot arms and the crane are allowed only to move if they are retracted and lifted, respectively.

Let's have a closer look into the controllers. There are three basic types of communication pattern according the order, in which input and output regions are acquired and released (see figure 4)[1)]:

1) Please note the following drawing convention. Shaded nodes are so-called **fusion nodes**. They serve as connectors: all fusion nodes with the same name are logically identical. Therefore, they will be merged physically for the analysis data structures. Usually, communication objects are represented by such fusion nodes to avoid immoderate edge crossing

Figure 4: Three types of cooperation pattern.



A. Independent input/output:
For the next operation step, the controller has to synchronize with only one of its adjacent controllers. E. g. to take a plate from the input area, a free output area is not required et vice versa. This pattern is applied to arms and crane.

B. Dependent input/output:
For the next operation cycle, the controller needs simultaneous control of input and output regions. This pattern is very useful to control the belts in such a way that the plates remain distinguishable. A belt is only switched on, if a new plate has been arrived (input available) and the output area is free. So at any time, maximal two plates can be on the belt - one at each end of the belt.

C. Mutually exclusive input/output:
At any time, the controller must hold a lock on one of its cooperation regions, i.e. the output region can only be released while having locked the input region and vice versa. This pattern is used for machines like table and press, which cannot be - at the same time - in a position suitable for loading as well as unloading.

Now let's consider the arms in more detail. They follow the independent input/output cooperation pattern. But additionally, both arms have to be synchronized in order to use the swivel only in a mutual exclusive manner. The two basic synchronization patterns have to be combined in an interleaving way.

Three possible arm versions are shown in figure 6. In version 1, the preconditions to start a motion step are acquired simultaneously. To implement this behaviour, corresponding compact language primitives are required which are usually not available in implementation languages. In opposite to that, arm version 2 and 3 correspond in a straightforward manner to the program sketches in figure 5 (taken from [4]).

Now we are ready to compose step by step the production cell's control system built from the machine components just introduced. The coarse structure given in figure 6 provides an overview of the whole (closed) process system. It shows the top level of an hierarchically structured Petri net which we get as result of the linking step. During linking, all (private) nodes of one process are uniquely prefixed to preserve node name uniqueness within the total system. Each of the macro transitions (represented as nested double boxes) includes the behaviour of one controller on the next lower level (i.e. the net structures of figure 4 or figure 6, but with prefixed node names).

Figure 5: Source text examples.

```
arm: procedure to take a plate

 Take /* version 2 */               Take /* version 3 */
 acquire lock on swivel;            acquire lock on input area;
 acquire lock on input area;        acquire lock on swivel;
    move_arm_to_grasppos;              move_arm_to_grasppos;
    do_grasp;                          do_grasp;
    go_in;                             go_in;
 release lock on input area;        release lock on input area;
 release lock on swivel;            release lock on swivel;
```

Figure 6: Three arm versions.

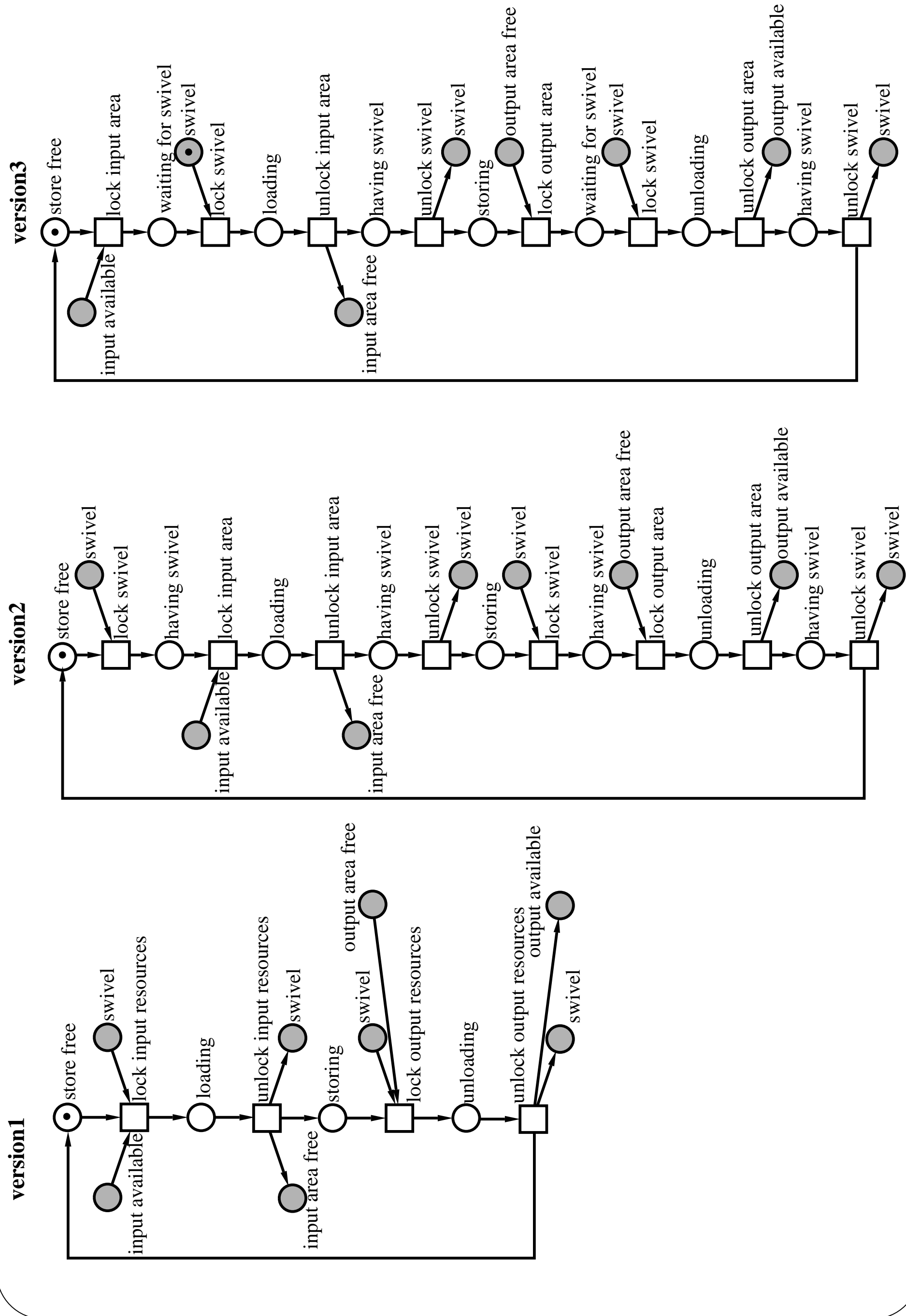
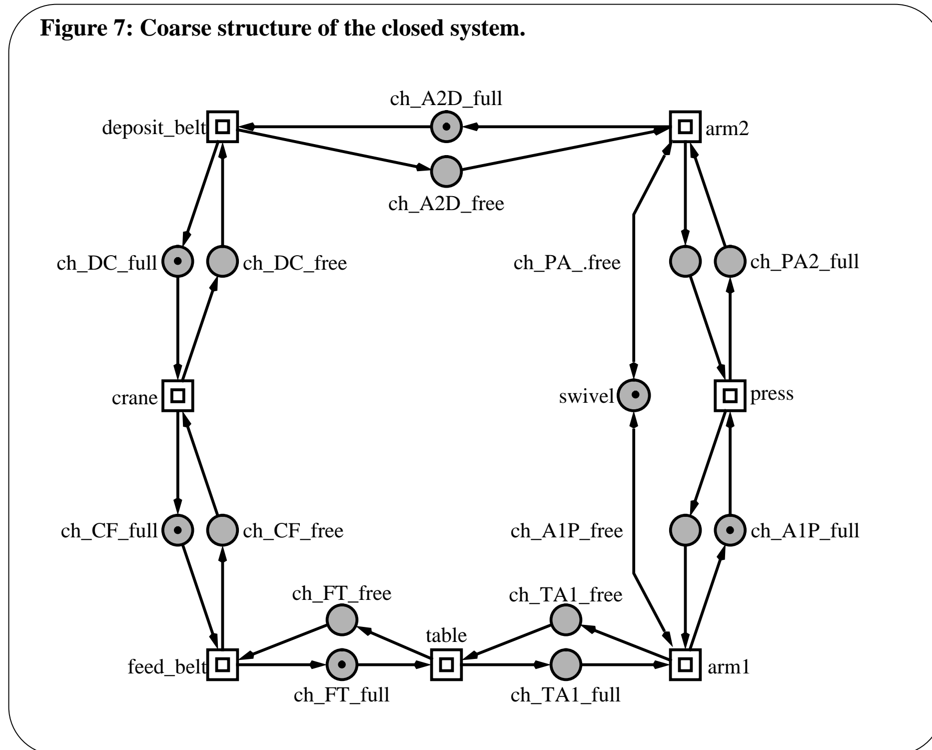


Figure 7: Coarse structure of the closed system.



3.3 Control Model

In order to be able to express safety requirements referring to physical devices, the controllers’ net models have to be extended by a net description of the environment reflecting all essential assumptions on its behaviour.

The environment model for each physical device is divided into two parts: an *actuator model*, which describes the possible states of a device and a *sensor model,* which expresses the sensor values which must be received by the controller (compare figure 8).

Actuators (e.g. the press’ engine) are effected by commands (e.g. press_upward, press_stop, press_down). We can identify the states of each device with the commands to control them (so for the press, there exist three possible states). A net description of the actuator states is obtained by adding a place P_A for each command A. A marking of P_A with one token means that the corresponding device has received the command A and is in an associated state.

A controller performs actions in response to specific (discrete) sensor values. To construct an environment model, describing the relations between sensors and actuators, it is enough to represent in the model only those finite discrete sensor values out of the

whole set of generally analogous values, which may cause a reaction of controllers (e.g. press_at_lower_pos, press_at_middle_pos, press_at_upper_pos). A net description of the relevant sensor states of the production cell is obtained by adding a place P_V for each of these values *V*. P_V is marked with one token if the value *V* is receivable by the control program.

Let's now discuss in more detail the model of the controller's interactions with the environment. Every complex control action is decomposed into elementary motion steps (like press_forge, press_lift etc.). One such step consists of device activation (start_command), waiting for a certain sensor value indicating that the motion has been completed (wait_stop_con(dition)), and device deactivation (stop_command), compare figure 8, above left.

A start command will force the associated device to change from an inactive to an active state. On the other hand, a stop command is assumed to force the device to change in an inactive state. The net in figure 8, above right (actuator state model) shows the modelling of these assumptions. The places stop_command and start_command represent the actuator states corresponding to the deactivation and activation of the actuator, respectively.

Each elementary motion step is performed in the context of an initial sensor value indicating the current position of the corresponding device. We assume that the performed motion will cause eventually the occurrence of a certain final sensor value. The places start_con(dition) and stop_con(dition) represent the initial and final values in the sensor state model. The transition css (change sensor state) implements this assumption (see above right, sensor state model).

To increase readability, control procedure and environment descriptions are abstracted by a macro component (coarse node with interface places) as shown in figure 8, below. Instantiating the macro net involves renaming of formal parameter places by actual parameters. The total net comprises 37 instances of this basic macro forming the sheets of the hierarchy tree.

4 Qualitative Analysis

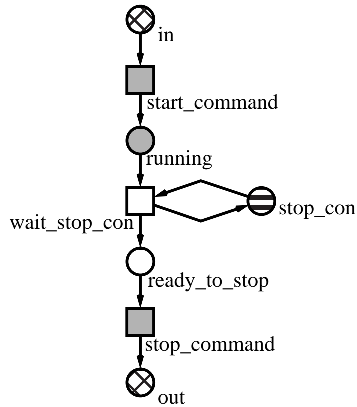
Due to the lack of applicable compositional approaches of Petri net analysis, all analysis results have to be confirmed after each refinement/composition step. But, it is a widely accepted engineers' basic principle that a sound composition/refinement has to be based on sound components. So, the successful analysis of a given model at a certain abstraction level is considered to be a necessary (but unfortunately not sufficient) condition to go ahead in modelling (compare figure 2). By this way, design faults have the chance to be detected early.

4.1 General Analysis

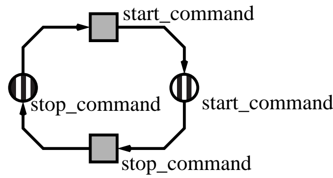
For the **cooperation model**, general analysis was done successfully using INA. Bound-

Figure 8: Petri net component of basic motion step and environment model.

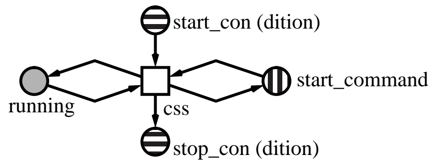
elementary motion step



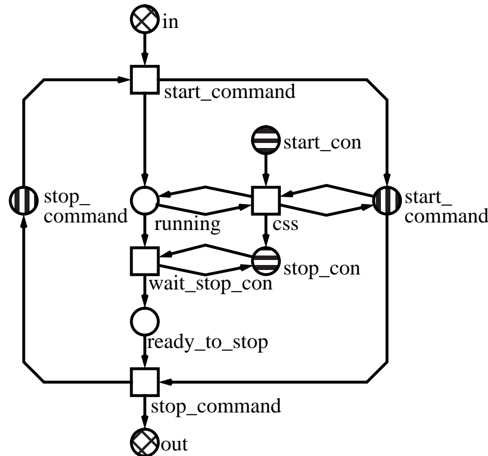
actuator state model



sensor state model

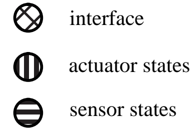


composition of the three models from above



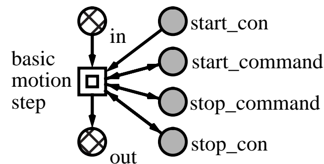
fusion nodes:

special fusion nodes:



css - change sensor state

macro component
(with formal parameters)



actual parameters, e.g.:

press_forge	press_lift
press_at_middle_pos	press_at_lower_pos
press_upward	press_up
press_stop	press_stop
press_at_upper_pos	press_at_middle_pos

edness and liveness could be decided efficiently (i.e. without construction of the complete reachability graph) by showing that the net is covered by semipositive place invariants and by proving the deadlock trap property (in connection with the net structure Extended Simple), respectively. Dead states caused by one discussed controller version (arm version 2) have been found very fast by construction of the (surprisingly small) stubborn set reduced reachability graph.

For the **control model,** boundedness is still decidable very fast by showing that the net is covered with semipositive place invariants. Due to the added environment behaviour, all net models exhibit a net structure beyond the Extended Simple one. So, the deadlock trap property could only show the freedom of dead states. But this can be proven more efficiently by constructing the stubborn set reduced reachability graph (even for the completely refined system with a still unknown state space size).

Because of the extraordinary size of the state space of the control model, the time and space effort to generate the complete reachability graph (to prove e.g. liveness) became unmanageable[1)].

Like many other “classical” net properties, the liveness property can also be expressed by a set of CTL formulae (one for each transition, see next section). But because their evaluation relies on the complete reachability graph, PROD’s CTL model checking component (which is based merely on graph traversing strategies and evaluation of state expressions), is not applicable in the case of the control model.

In opposite to that, the evaluation of a LTL formulae may be based on a stubborn set reduced reachability graph, resulting generally to much smaller sets of reachable states. But in LTL, only a stronger liveness property of transitions can be expressed, which is related to the livelock freedom of transitions. Obviously, every livelock-free transition is also live. We have checked this formula for every transition in the control model using a batch program. But, the formula does not hold for all transitions in the control model (-as expected, any actions in alternative execution branches are not livelock-free). For the remaining transitions we succeeded to prove liveness using the model checker of the PEP tool (see next section for a comparison of the expressive power of the temporal logics supported by the different tools).

4.2 Special Analysis

To highlight differences between the applied tools concerning their expressiveness, it is useful to summarize typical questions/properties dealt with during special analysis. In the following φ stands shortly for a general logical expression characterizing usually a (wanted or unwanted) state or set of states.

1) We stopped the state space construction after about two days, having generated about 700 000 states filling about 1.5 GigaBytes.

(1) **reachability-related** (reachability of a state where φ holds):

$$\mathbf{EF}\ \varphi$$

(There exists at least one computation path (future behaviour) to reach eventually a state where φ will be true.)

(2) **safety-related** (unreachability of a state where φ holds):

$$\mathbf{AG}\ (\neg\varphi), \qquad \text{(equivalent to } \neg\mathbf{EF}\ \varphi)$$

(For every computation path, φ will never be true.)

(3) **invariant-related** (general validity of an assertion φ):

$$\mathbf{AG}\ \varphi, \qquad \text{(equivalent to } \neg\mathbf{EF}\ (\neg\varphi))$$

(For every computation path, φ will be true for ever.)

(4) **liveness-related**:

$$\mathbf{AG}\ \mathbf{EF}\ \varphi$$

(What ever happens, there exists the chance (at least one path) that φ will be true.)

(5) **progress-related**:

$$\mathbf{AG}\ \mathbf{AF}\ \varphi$$

(For every computation path, φ will eventually be true.)

Which tools *may* be applied at all for a given type of question depends on the temporal logic it provides:

Tool	Supported type of logic	Operators	Type of properties expressible
INA	-	**EF** φ (but φ can only be given by a (sub-) marking)	(1), (2)
PEP	(restricted) CTL	**AG**, **EF**	(1) - (4)
PROD	LTL (without next-time operator)	**G**, **F**, **U** (unquantorized versions of **AG**, **AF**, **AU**)	(2), (3), (5)
PROD	(full) CTL	**EX**/**AX**, **EF**/**AF**, **EG**/**AG**, **EU**/**AU**	(1) - (5)

Which tools *should* be applied in which order depends on the analysis methods they are based on.

The analyses of PEP are based on a so-called finite prefix of branching processes [11] [12] [21]. In case of concurrent systems, these graphs are much smaller as the “classical” complete reachability graphs because they alleviate the state explosion by avoiding the enumeration of all interleaving combinations for independently concurrent actions.

INA as well as PROD/LTL use stubborn set reduced reachability graphs (but in a different way), which are generally, in case of highly concurrent systems, surprisingly far smaller than the complete reachability graph. Such a reduced (very small) graph is constructed newly for each question.

We succeeded in construction of stubborn set reduced versions of the reachability graph as well as of the finite prefix of branching processes even for systems which total system state size we don’t know.

Because the evaluation of PROD’s CTL relies on the complete reachability graph, its application should only be tried, if the formerly mentioned methods do not help.

Obviously, this helpful meta-knowledge on the tool-internal analysis techniques should be provided to the tool box’ users, e.g. by a dialogue-oriented user guideline of the Petri net framework implementation (see figure 1).

So, safety (and other) requirements of the cooperation model expressed by CTL and requirements of the control model expressed by LTL have been proven successfully. To demonstrate the variety, let’s give some examples.

(1) reachability-related, e.g.:
To gain deeper insight into the controllers’ concurrency: *Is it possible that both robot arms hold a plate at the same time?*

$$\mathbf{EF}\ (arm1_mag_on \wedge arm2_mag_on)$$

(2) safety-related, e.g.
The press may only be closed, if no robot arm is positioned inside it, i.e. for arm 1:

$$\mathbf{G}\ \big((arm1_release_angle \wedge arm1_release_ext) \rightarrow (press_stop \wedge \neg press_at_upper_pos) \big)$$

To avoid machine collisions, the robot may only rotate, if both arms are retracted:

$$\mathbf{G}\ \big((robot_left \vee robot_right) \rightarrow (arm1_retract_ext \wedge arm2_retract_ext) \big)$$

(3) invariant-related, to prove design consistency, e.g.
The press is either stopped or moves in exactly one direction, i.e. is always in one of its actuator states:

$$\mathbf{G}\left(press_stop \ \dot{\vee}\ press_upward \ \dot{\vee}\ press_down \right)$$

The press is always positioned (logically) at exactly one of its sensor states:

$$\mathbf{G}\left(press_at_lower_pos \ \dot{\vee}\ press_at_middle_pos \ \dot{\vee}\ press_at_upper_pos \right)$$

(4) liveness-related, e.g.
A Petri net transition t is live iff it may be enabled infinitely often:

$$\mathbf{AG\ EF} \bigwedge_{p \in {}^{\bullet}t} \tilde{p}$$

(A transition t of an ordinary Petri net is enabled (may fire) if all its preplaces ($p \in {}^{\bullet}t$) hold a token (here $\tilde{p}$ denotes the interpretation of a place name as an atomic proposition: $\tilde{p}$ yields true if p is marked with one token at a state where the proposition is evaluated).

(5) progress-related, e.g.

$$\mathbf{AG\ AF} \bigwedge_{p \in {}^{\bullet}t} \tilde{p}$$

A transition t is livelock-free iff it will be enabled infinitely often:

PROD's evaluation method of LTL formulae has been proven applicable even for medium-sized systems.[1)] However, liveness-related properties cannot be expressed in LTL because of the lack of quantification on computation paths. On the other hand, PROD's model checker for full CTL formulae is not applicable for the control model because it depends on the complete construction of it's state space. Surprisingly good results are gained by using PEP's model checking algorithm for the verification of liveness-related as well as safety-related properties[2)]. Therefore, the model checking techniques provided by PEP and PROD seems to be complementary to each other.

5 Synthesis

To avoid any additional implementation faults, the actual control software has been directly synthesized from the Petri net specification. Based on the FUN Petri net simulator [25], we have automatically generated a FUN description of the total net structure and a C-procedure skeleton for each transition. There are 37 basic macro transitions, containing the elementary motion steps. For the three transitions of all these basic macros (start_command, wait_stop_con, stop_command, see figure 8, middle) the corresponding procedure skeleton had to be filled with the actual elementary motion code. The remaining transitions simply play the token game. The assignment of all these procedures to the corresponding transitions is handled by name equivalence. The procedures are executed if the corresponding transition fires. All elementary motion code declarations are local to their C-procedure skeletons. This prevents destruction of the

---

1) The sizes of the stubborn set reduced reachability graphs constructed to evaluate formulae like those given above have been between 500 and 30.000.

2) Liveness of transitions, for instance, can be checked in an insignificant amount of time (0.04 seconds)! Similar results are obtained for simple safety formulae of type (2) (see above). The finite prefix of the control model consists of 1619 conditions and 768 events, constructed in less than 0.1 seconds on a SPARC Station 20.

Figure 9: FUN transition code.

```
void A1L_ext_Pstart(FunParamBlock & In, FunParamBlock & Out,
                    FunParamBlock & SigIn, FunParamBlock & OutSig )
{
/* pre-places */
    long A1Lrotated =  Value(In[0],long);
    long arm1_stop =  Value(In[1],long);
/* post-places */
    long A1L_ext_run =  Value(In[0],long);
    long arm1_forward =  Value(In[0],long);
/* transition code begin */
    // Extension_Of_Arm1(Arm1_Retract_Extension)
    //         -> Extension_Of_Arm1(Arm1_Pickup_Extension)

    cout << "arm1_forward" << endl;
/* transition code end */
    Out[0] = new FunInteger(A1L_ext_run);
    Out[1] = new FunInteger(arm1_forward);
}
```

well-analysed net behaviour. Therefore, any (implicit) communication between these procedures has been made impossible.

The generated control software runs in a simulation environment of the production cell implemented with Tcl/Tk. The communication between the control software and the simulation environment is based on a simple Input/Output protocol. Therefore, the elementary motion code consists of simple I/O statements. The example in figure 9 illustrates the syntactical structure of an elementary motion step procedure (corresponding to the start_command of the basic macro transition to extent arm 1 from the retract position to the pick-up position). There are two parts. One represents the automatically generated procedure skeleton (plain), and the other contains the included elementary motion code (bold).

6 Conclusions

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:

- Worst-case evaluation by Duration Interval Nets (firing of transitions consumes time characterized by interval delays) [19] to prove the meeting of given deadlines (implemented in the latest update of INA).
- Quantitative analysis by Stochastic Nets [33] for performance and reliability evaluation.

- Incorporation of fault tolerance aspects (blowing up the net sizes significantly).

Throughout this paper, (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.

Finally, the main lessons learnt concerning a suitable tool box framework are the following.

- The combination of different tools (even if they provide similar features at the first glance) seems to be unavoidable.
- We need user guidelines showing which analysis techniques are recommendable for a given analysis question.
- The check of a given system against its functional and/or safety requirements given by a (more or less large) set of temporal formulae calls for distributed evaluations in batch processing manner.

References

[1] BALBO, G.: Performance Issues in Parallel Programming; LNCS 616, 1992, pp. 1-23.

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

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

[4] 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.

[5] CHANG, C. K. ET AL.: Integral: Petri Net Approach to Distributed Software Development; Information and Software Technology 31(89)10, pp. 535-545.

[6] 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(86)2, pp. 244-263.

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

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

[9] DONATELLI, S. ET AL.: Use of GSPNs for Concurrent Software Validation in EPOCA; Information and Software Technology 36(94)7, pp. 443-448.

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

[11] ENGELFRIET, J.: Branching Processes of Petri Nets, Acta. Inf., 25(91), pp. 575-591.

[12] ESPARZA, J.: Model Checking Using Net Unfoldings, Science of Computer Programming, 23(94), pp. 151-195.

[13] 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, pp. 3-18.

[14] GERMAN, R. ET AL.: TimeNet - A Tool Kit for Evaluating Non-Markovian Stochastic Petri Nets; Techn. Univ. Berlin, Dep. of CS, Report 1994-19.

[15] HEINER, M., Petri Net Based Software Validation, Prospects and Limitations, ICSI-TR-92-022, Berkeley/CA, 3/1992.

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

[17] HEINER, M.: Petri Net Based Software Dependability Engineering; Tutorial Notes, Int. Symposium on Software Reliability Engineering (ISSRE '95), Toulouse, Oct. 1995.

[18] HEINER, M., DEUSSEN, P.: Petri Net Based Qualitative Analysis - a Case Study; BTU Cottbus, Dep. of CS, Techn. Report I-08/1995.

[19] HEINER, M., POPOVA-ZEUGMANN, P.: Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets; BTU Cottbus, Dep. of CS, Techn. Report I-02/1996.

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

[21] MACMILLAN, K. L.: 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, pp. 164-174.

[22] MENZEL, T.: Design and Implementation of a Petri Net Tool Kit Framework Integrating Animation and Simulation (in German); BTU Cottbus, Dep. of CS, Internal Manuscript 3/1996.

[23] POPOVA, L.: On Time Petri Nets; J. Information Processing and Cybernetics EIK 27(91)4, pp. 227-244.

[24] ROCA, J. L.: A Method for Microprocessor Software Reliability Prediction; IEEE Trans. on Reliability 37(88)1, pp. 88-91.

[25] 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.

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

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

[28] STARKE, P.: A Memo On Time Constraints in Petri Nets; Humboldt-University zu Berlin, Informatik-Bericht Nr. 46, August 1995.

[29] VALMARI, A.:A Stubborn Attack on State Explosion, Formal Methods in System Design 1(92)4, pp. 297-322.

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

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

[32] VARPAANIEMI, K. ET AL.: PROD Reference Manual; Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, August 1995.

[33] 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.