USING EXTENDED PREDICATE/TRANSITION-NETS FOR THE SPECIFICATION, ANALYSIS AND SYNTHESIS OF EMBEDDED REAL-TIME SYSTEMS

Jürgen Tacken[1]

1. Introduction

In recent years embedded systems have gained increasing importance. Due to the increasing functionality they have to be designed in teams with several specialists, each of them working on one single part of the whole system. But the focus in design is not only more functionality and higher performance but also safety and reliability criteria that have to be fulfilled by the designed components. This includes functional requirements as well as real-time constraints.

By now every single area of application has developed its own well understood techniques for modeling, combined with corresponding methods and tools for analysis and simulation. Already at the state of many individual models many predictions about the temporal and functional behavior of each subsystem can be made. But to validate the behavior of the whole system, the individual models have to be coupled. Especially for the systems' reliability it is important to consider not only each single component on its own but its behavior within the whole context. Many functional errors only expose themselves when all individual components work together in the whole context, observed over time.

Often the models are given in different domain-specific modeling languages, so coupling can only be realized either on the level of simulation or within a hybrid language, that usually does not offer any facilities for further analysis. Coupling of simulators allows a hybrid simulation in the sense of a combined simulation of all subsystems, while each subsystem may be formulated in a different language for a specific simulator. On the level of simulation temporal and functional behavior and

[1] C-LAB, Fürstenalle 11, 33094 Paderborn, Germany, Email: theo@c-lab.de

performance can be studied and validated. But not all errors can be found by simulation because of the exhaustive number of possible simulation runs. Hence it is desirable to run a formal analysis of the static and dynamic properties for formal verification purposes.

A further problem in a separated design process of different subsystems is that each individual domain has proprietary methods of optimization, but for a global optimization of the joint system no facility exists. Especially when coupling different subsystems it may be useful and more cost-effective to export some functionality from one into another subsystem. An overall analysis of the joint system may reveal states that can never be reached and therefore can be eliminated from the design. The prerequisite for this kind of analysis is that all models are given in an uniform language.

In this paper we will describe a new method for the design of embedded real-time systems based on a design flow. During the specification and modeling phase of the design flow it allows the use of several domain-specific modeling languages. All these different languages are transformed into one common model using only one modeling language. So this modeling language has to be very powerful since it must have at least the functionality of every single domain-specific modeling language. After combining all the different designs a global analysis and formal verification of the system may be performed on the common model. To apply efficient and useful analysis and verification methods together with a corresponding software and hardware synthesis the common modeling language is transformed into a less powerful modeling language.

In Chapter 2 we describe our common formal model with all its extensions useful for the integration of different modeling languages. In Chapter 3 we outline our design flow. Chapter 4 deals with the transformation of the powerful model into less powerful versions. Together with these transformations their effect on the analysis results is discussed. In Chapter 5 a timing analysis method for the common model is introduced. The timing analysis results in a partitioning of the system guaranteeing that the real time restrictions are met. This partitioning is the basis for the synthesis into distributed real time software and special hardware. Chapter 6 summarizes the results presented in this paper.

2. Extended Predicate/Transition-Nets

When modeling embedded systems it is important to handle concurrency. As shown in (Dittrich, 94) petri nets are very well suited for the specification of concurrent systems. To handle the complexity they use hierarchical petri nets as specification language in order to support hardware/software-codesign. In our work we also decided to use an extended form of petri nets, namely extended Predicate/Transition-Nets (extended Pr/T-Nets). They have a well defined semantics that allows modular specifications of parallel behavior. Additionally the integration of other specification languages is supported and the representation of the internal model in different specific views is possible. Pr/T-Nets (Genrich, 1987) are bipartite graphs consisting of *places* and *transitions*. In order to define dynamic behavior the places may contain *tokens*. The actual set of all tokens on all places is called a marking of the net. The *edges* between places and transitions (partly) define the possible flow of tokens in a net by the so called *firing* of transitions. To further specify the flow in Pr/T-Nets edges may be annotated by sums of variable tuples. Transitions carry first order formulae over a set of constants and variables together with expressions used to calculate values for variables occurring at output edges of a transition. Transitions are *enabled* (can fire) if appropriate tokens are available on all their input places. When a transition fires it removes tokens from its input places and produces some new tokens on its output places according to the flow specified by the edges and the annotations. In this way the behavior of the Pr/T-Nets is formally defined.

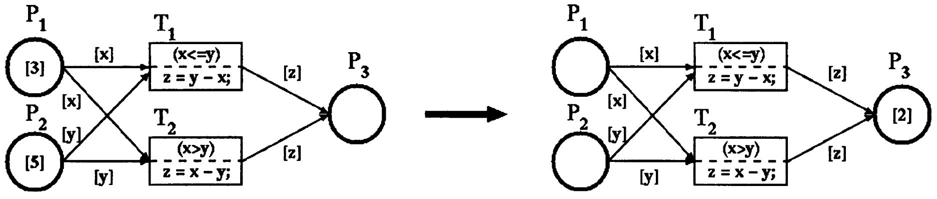

Figure 1: Pr/T-Net example

Figure 1 shows a simple Pr/T-Net calculating the distance of two integers. The places P_1 and P_2 have to be marked with tokens containing the arguments for the calculation. Within the marking shown on the left a substitution of the variables *x* by *3* and *y* by *5* determines a valid set of tokens for which the transition T_1 may fire, since $3 <= 5$. The right part of Figure 1 shows the net after firing of T_1.

We extended the basic definition of Pr/T-Nets with hierarchy and timing concepts. Hierarchical specifications are useful to handle complexity in large designs, support a modular specification, and allow the reuse of predefined nets in several models. Our hierarchical Pr/T-Net-model allows the refinement of places and transitions by subnets. Nodes refined by subnets are called structured nodes. The subnet of a structured node is itself an extended Pr/T-Net which may again contain structured nodes. A subnet of a structured node may have connections to nodes in the surrounding net. The connections have to be made via special nodes of the subnet, that are called ports. The net depicted in Figure 1 defines three ports for the inputs and the output of the calculation. Ports are marked by a bold outline. We allow several modes for the semantics of a structured node. In general the semantics is defined via the activity of the subnet. A subnet of a structured transition is active as long as the structured transition itself is active. This is similar to the philosophy of structured nets described in (Cherkasova, 1981). The subnet of a structured place is active as long as the structured place contains at least one token. The concept for structured places is similar to the hierarchical concept in statecharts. The most simple case is a structured place that is initially marked and has no connections to other nodes. Its semantics complies to replacing the node by its refining subnet.

The definition of hierarchical Pr/T-Nets is closely coupled with a mechanism for graphical abstraction of Pr/T-subnets. Based on this graphical abstraction a domain specific representation of the internal model is possible. It provides the ability to define an intuitively understandable abstract graphical representation that is also able to continuously represent the system's behavior and state changes during simulation. The description of the abstract representation may use arbitrary graphical elements. Hence, existing graphical specification languages can be (re)produced by using their predefined symbols as the abstract representation of the petri net elements (Kleinjohann, 1996/1997). Furthermore, an engineer can define his own graphical representation of the system in a way he is familiar with.

Our timing concept supports the integration of real-time aspects like execution and idle times for actions into a Pr/T-Net-model. For each transition an interval or a single integer value may be defined as enabling delay and firing delay. The enabling delay determines the time units before a transition may become active after it has been enabled and the firing delay specifies how long a transition is active. If a transition is active, the tokens from the input places are removed but the tokens for the output places are not yet produced. With the enabling delay idle or waiting times can be modeled. The firing delay is used to specify execution times of the annotations. In addition to the time definition for the transitions a function is needed, mapping the integer time units to real physical times.

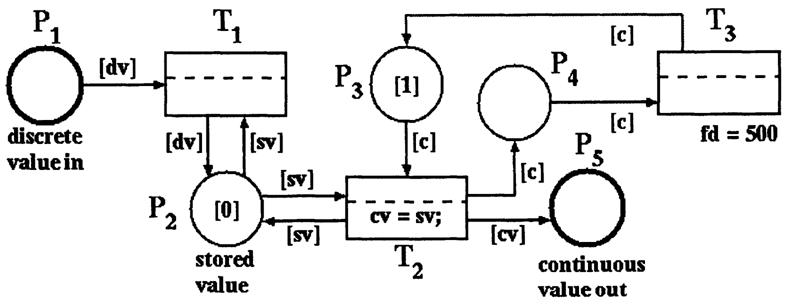

Figure 2: Timed Pr/T-Net example

In Figure 2 an example for a Pr/T-Net with timing specifications is depicted. The net is a typical example for an element used for interfacing models of different system parts, e.g. of a discrete event driven part and a periodic one, that was originally specified by a continuous model. Each value arriving aperiodically at the in-port of the net is stored in place P_2 by transition T_1. The stored value is sent to the out-port by transition T_2. Due to the firing delay defined for transition T_3 the value is sent with a period of 500 time units. If for example the integer time unit is mapped to the physical unit μs, the value would be produced with a frequency of 2 kHz.

For a comfortable specification, simulation, and animation of extended Pr/T-Nets we use the System Engineering and Animation-environment SEA (Kleinjohann, 1996).

3. Design flow

To advance towards a complete design method one has to take the flow of design into account. The process we suggest in this work is divided into the three stages modeling, analysis and synthesis (cf. Figure 3).

3.1 Modeling

The process of modeling usually starts from a concept of a given system the designer has in mind. During a manual prepartitioning phase he roughly defines which components will constitute the system. For the application described here this should be a partitioning into software, controllers having continuous behavior, and digital hardware. This prepartitioning is based upon the experience of the engineer developing the system.

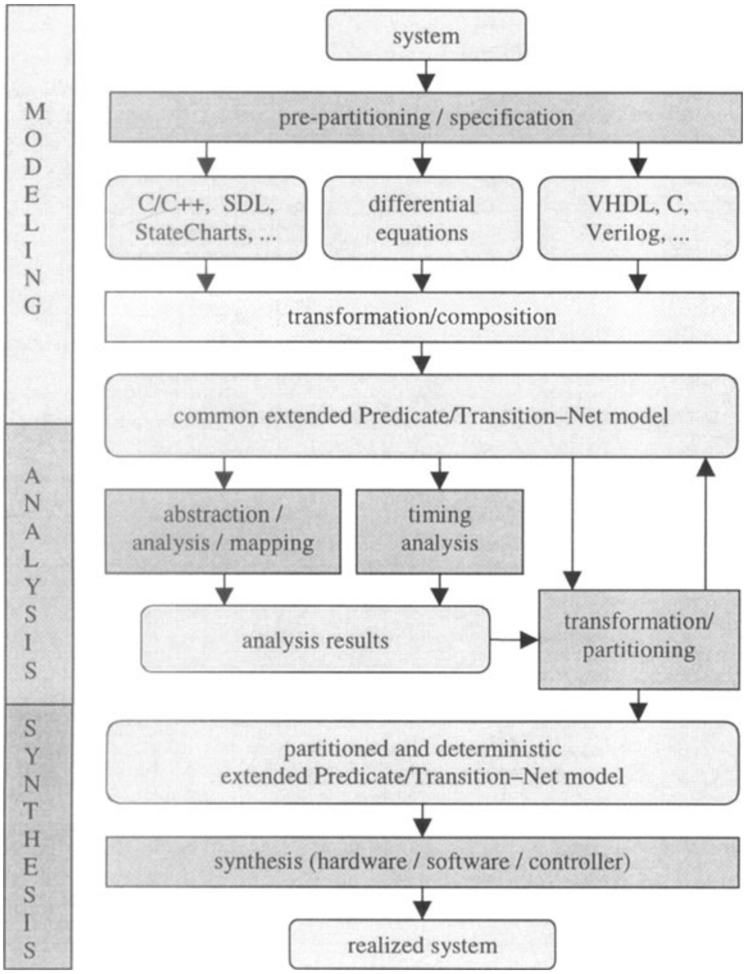
MODELING
ANALYSIS
SYNTHESIS
system
pre-partitioning / specification
C/C++, SDL, StateCharts, ...
differential equations
VHDL, C, Verilog, ...
transformation/composition
common extended Predicate/Transition–Net model
abstraction / analysis / mapping
timing analysis
analysis results
transformation/ partitioning
partitioned and deterministic extended Predicate/Transition–Net model
synthesis (hardware / software / controller)
realized system

Figure 3: Design flow with the three stages modeling, analysis and synthesis

Typically designers of the individual components choose a specification language they are familiar with and that is appropriate for the area of application. Generally these languages have their own domain-specific tools for analysis and optimization which are not subject of this work. There are two possible ways to transform models given in different specification languages into an extended Pr/T-Net model.Using transformation each individual specification can be transformed, either manually or automatically, into an extended Pr/T-Net. Depending on how skillful this transformation was performed a more or less complex Pr/T-Net is created that realizes the specification. A transformation for differential equations is presented in (Brielmann, 1995). An example for the transformation of SDL (Specification and Description Language) is shown in (Kleinjohann, 1997).

Another method for the SEA-environment is a library construction. The process of modeling may be performed using components from a library directly within the graphical SEA-environment. For all components there is an underlying Pr/T-Net that can be combined with other components, resulting in an executable specification. So a complete hierarchical Pr/T-Net is generated automatically. Of course, modeling this way the resulting net will be more complex than the one generated by a direct transformation of the specification. However, the development of a transformation is much more costly than the creation of a library of language components. For the SEA-environment we have evaluated this library method for the specification of data flow, block diagrams for differential equations, and asynchronous hardware on gate level, but other graphical languages may be supported, too.

All resulting Pr/T-Nets now have to be coupled into a combined net, using an interface model library for coupling the subnets together. There are a number of interface models for all combinations of different subnets from different domains. These interface models are e. g. necessary to combine the different modeling techniques of event-based and continuous system parts. The combined net can be modified manually with the help of a graphical editor. At this stage of the design process we are able to validate the specification by a very important feature of the SEA-environment. With the built in Pr/T-Net simulator and -animator the specification can be executed and tested. The readability of the simulation is supported by an animator which shows an abstract graphic representation of the underlying execution.

3.2 Analysis

In order to allow the integration of several different specification languages the extended Pr/T-Nets as common specification language are a very powerful language. The problem with such powerful modeling languages is, that they are very hard to analyze even if they are formal languages like Pr/T-Nets. But analysis is very important for the systems reliability and to support an effective synthesis of the system. Hence we try to abstract from the complex model of extended Pr/T-Nets to simpler petri net models. For such models a lot of tools for analysis exist and may be used. Afterwards we try to map the results to the original Pr/T-Net model. This is explained in more detail in section 4.

Another aspect of analysis is to do timing analysis, for example to determine the different execution times for alternative realizations of functions for different target architectures. Such alternative realizations can be specified using non determinism. Each alternative may have its advantages or disadvantages for different target architectures or target code languages. The design method should then support an automatical selection of the best alternative for synthesis. This is explained in more detail in section 5. The results of this analysis can either be used for the partitioning or for the synthesis phase. The timing results are for example used to transform non deterministic subnets into deterministic ones.

3.3 Synthesis

After a stable state is reached in the alternating phases of modeling and analysis, the partitioned net graph can be compiled into hardware and real-time software during the phase of synthesis. The compilation process can be supported by results of the analysis. In the case of digital hardware the net is transformed into RT-level VHDL code using the Paderborn MOdular System for High-Level Synthesis and HW/SW-codeSign (PMOSS) (Hardt, 1995). In the case of distributed real-time software one compiles the corresponding Pr/T-subnet using the C-LAB Hard Real-Time System (CHaRy) (Altenbernd, 1997). These tools are both parts of the PARADISE Design Environment (Hardt, 1998).

4. Analysis for the verification of system properties

As described above, we made various extensions to the formal model of Pr/T-Nets in order to meet all requirements for a common modeling language. Unfortunately, these extensions have the drawback that all the well-known analysis methods for petri-net are not applicable to our models. We decided to use existing petri net methods for the analysis phase of our design method. The main properties for the analysis of petri nets are boundness, reachability, deadlock-freeness and liveness (Starke 1990). In the following we describe these properties and their meaning for the modeled embedded systems.

A place is bound if a positive number exist which is an upper limit for the number of tokens on this place for every possible firing sequence within the net. If this upper limit is equal to 1 the place is called safe. If all places are bound/safe than the whole net is bound/safe. If places are unbounded in the model there may exist potentially overflowing buffers or infinite process calls in the realization of the embedded system.

A marking in a net is called reachable if a firing sequence of transitions exists leading from the initial marking to this marking. A marking in the net represents a state of the modeled system. Hence knowing if a marking is reachable means to know if the corresponding state of the embedded system is reachable. With this property one can for example check if in case of emergency conditions a safe state is reached.

A marking is called a deadlock if no transition of the net is enabled for this marking. So there is no further action possible within the modeled system which is often not intended. In case of a deadlock every transition of the net is dead. A transition is called live for a marking *M* if it is not dead for *M* and also for all markings that are reachable from *M*. If all transitions are live for a marking then the marking is called live and if all reachable markings of a net are live the net is called life. If a net is live then it can not have any deadlocks.

Some properties can already be checked with a structural analysis of the net graph. This can be done directly for the Pr/T-Net graph. For example if there exists a transition without input places it is live and all output places are unbounded or if a place has no input transitions it is bounded and all output transitions are live. If *t* is a transition without output places then *t* is not live or all input places of *t* are unbounded and if *p* is a place without output transitions then *p* is unbounded or all input transitions of *p* are not live. More important than these structural results are results of dynamic analysis like for example occurrence graph based analysis. In the occurrence graph of a net a node represents a state in the net. A state is characterized by the actual marking of places with token. This is represented by a tuple containing the set of token for every place.

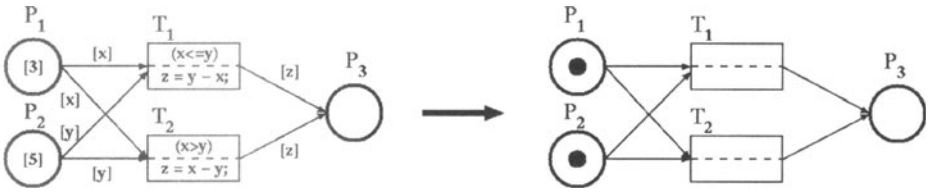

Figure 4: Simple abstraction of an extended Pr/T-Net

To use the occurrence graph based analysis methods for extended Pr/T-Nets we need an abstraction from extended Pr/T-Net models to petri net models. Figure 4 shows an example for a very simple abstraction. If we ignore all the annotations of transitions, edges and token the result is a simple petri net. This has of course effects on the resulting occurrence graph.

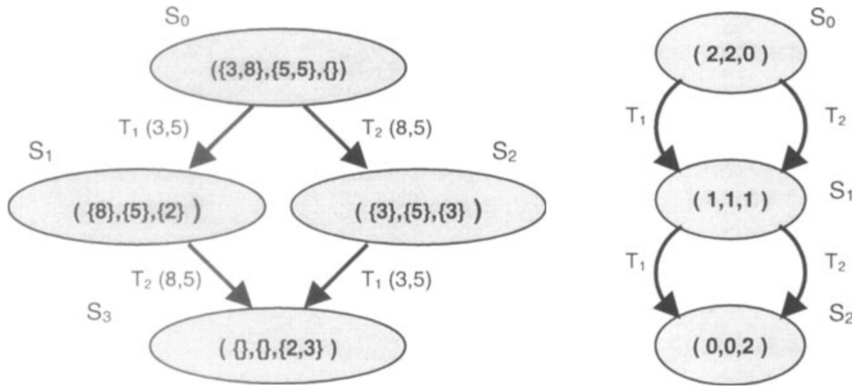

Figure 5: Occurrence graphs for Pr/T-Net and abstracted petri net

In the left side of Figure 5 the occurrence graph for the Pr/T-Net from Figure 4 is depicted. The first state in this graph indicates that P_1 is marked with 3 and 8, P_2 with 5 and 5, and P_3 contains no token. The edges are annotated with the transition, that causes the change of the state and the chosen variable substitution. For example if T_1 fires with the substitution 3 and 5 the state S_1 is reached. The occurrence graph for the abstracted net depicted on the right side of Figure 5 contains less nodes, because the states are not distinct by individual token. Only the number of token on a place is relevant. But if you examine the possible firing sequences of transitions, you get more than in the Pr/T-Net. For example in the petri net the transition T_1 may fire twice but not in the Pr/T-Net. This means the annotations in the Pr/T-Net define a restriction on the token flow.

Based on this fact some results from the analysis of the abstracted nets can directly be mapped to the Pr/T-Nets. If for example a marking is not reachable in the petri net it is also not reachable in the corresponding Pr/T-Net. But if it is reachable in the petri net it is not necessarily reachable in the Pr/T-Net. But of course we can use the petri net analysis to determine all possible firing sequences leading to a marking and than just check if one of these firing sequences also exists in the Pr/T-Net. If a deadlock occurs in the abstracted net it is also present in the Pr/T-Net if the corresponding marking is reachable. But not having any deadlocks in the abstracted net does not mean the Pr/T-Net is deadlock free. For safeness or boundness it holds that the Pr/T-Net is bound/safe if the abstracted petri net is bound/safe. But if the abstracted net is unbounded/unsafe the Pr/T-Net may be bound/safe.

5. Timing analysis

To support a software/software partitioning of the net and the elimination of non determinism we perform a timing analysis (Rust, 1998). As a first step we calculate execution times for every transition annotation in the examined net part using the C–LAB Hard Real-Time System CHaRy (Altenbernd, 1997). CHaRy determines a worst case execution time for each annotation on a specified target architecture. This worst case execution time is used to annotate the transitions in the Pr/T-Net with delay values. Afterwards a reachability analysis is done to determine all possible firing sequences leading from an input to the desired output and a rough worst case execution time for all firing sequences can be calculated. Now the best fitting alternative can be chosen from all alternatives.

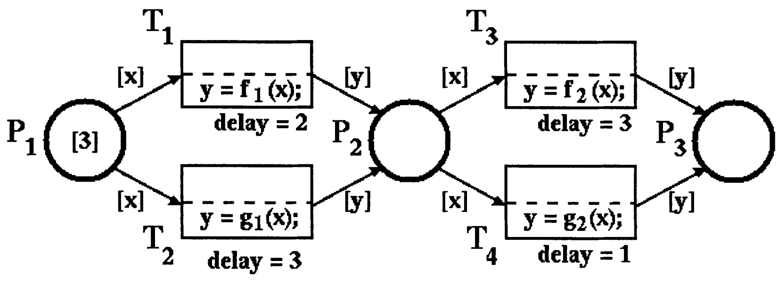

Figure 6: Pr/T-Net with calculated worst case execution times

Figure 6 shows an example for an extended Pr/T-Net with alternative firing sequences and calculated worst case execution times for each transition. The transitions T_1 and T_2 are in conflict. If P_1 is marked each of them may fire. Which one fires is completely non deterministic. So this is a kind of design freeness. Each of them produces a correct result but they have different execution times on the specified target architecture. The same holds for the transitions T_3 and T_4. In this case the best result is achieved for the sequence T_1,T_4 and the worst for T_2,T_3. So the transitions T_2 and T_3

can be eliminated from the model. For the remaining transitions the delay values are used to perform a partitioning on the extended Pr/T-Net model.

6. Conclusion

In this paper we described a new method for the design of embedded real-time systems. Our suggested design flow is an approach towards an integrated, complete design. We have shown how to use an extended form of Pr/T-Nets as a common model for specification. We explained how to use standard petri net analysis methods to achieve results for extended Pr/T-Nets. These results are used for the verification of system properties. Additionally we introduced timing analysis to determine the different execution times for alternative realizations of functions for different target architectures and to support the partitioning of the modeled system. This partitioning is the basis for the synthesis of the system.

References

Altenbernd, P. (1997). CHaRy: The C-LAB Hard Real-Time System to Support Mechatronical Design in International Conference and Workshop on Engineering of Computer Based Systems (ECBS), Monterey, CA.

Brielmann, M. (1995). *Modelling differential equations by basic information technology means.* Proceedings of the 5th International Conference on Computer Aided Systems, Theory and Technology (EUROCAST'95), Innsbruck, Austria.

Cherkasova L. A., Kotov V. E. (1981). *Structured nets.* In J. Gruska and M. Chytil, editors, Mathematical Foundations of Computer Science, volume 118 of Lecture Notes in Computer Science. Springer Verlag.

Dittrich, Gisbert. (1994). *Modeling of Complex Systems Using Hierarchical Petri Nets.* In Codesign: Computer-aided software/hardware engineering, New York, NY: IEEE Press, chapter 6.

Genrich, H. J. (1987). *Predicate/Transition Nets.* Advances in Petri Nets Part I, Vol. 254, Springer Verlag.

Hardt, W. (1995). An Automated Approach to HW/SW-Codesign. In IEEE Colloquium: Paritioning in Hardware-Software Codesings. London, Great Britain, February 1995.

Hardt, W., Altenbernd, P., Böke, C., Del Castillo, G., Ditze, C., Erpenbach, E., Glässer, U., Kleinjohann, B., Lehrenfeld, G., Rammig, F. J., Rust, C., Stappert, F., Stroop, J., Tacken, J. (1998). "Paradise: Design Environment for Parallel & Distributed, Embedded Real-Time Systems", In Proc. of the International IFIP WG 10.3 / WG 10.5 Workshop on Distributed and Parallel Embedded Systems (DIPES '98), Paderborn University, Germany.

Kleinjohann, B., Kleinjohann, E., Tacken, J. (1996). *The SEA Language for System Engineering and Animation.* Applications and Theory of Petri Nets, LNCS 1091, Springer Verlag.

Kleinjohann, B., Tacken, J., Tahedl, C. (1997). *Towards a Complete Design Method for Embedded Systems Using Predicate/Transition-Nets.* In Proc. of the XIII IFIP WG 10.5 Conference on Computer Hardware Description Languages and Their Applications, Toledo, Spain, Chapman & Hall.

Rust, C., Stroop, J., Tacken, J. (1998). "The Design of Embedded Real-Time Systems using the SEA Environment", In Proc. of the 5th Annual Australasian Conference on Parallel And Real-Time Systems (PART '98), Adelaide, Australia.

Starke, P. H. (1990). Analyse von Petri-Netz-Modellen, Leitfäden und Monographien der Informatik, Teubner, Stuttgart.