2

Towards a Complete Design Method for Embedded Systems Using Predicate/Transition–Nets*

B. Kleinjohann
C-LAB
Fürstenallee 11, 33102 Paderborn, Germany, bernd@c-lab.de

J. Tacken
C-LAB
Fürstenallee 11, 33102 Paderborn, Germany, theo@c-lab.de

C. Tahedl
Heinz Nixdorf Institut
Fürstenallee 11, 33102 Paderborn, Germany, tahedl@uni-paderborn.de

Abstract

In this paper, we present a new approach to embedded system design based on modeling discrete and also continuous system parts with high level Petri–Nets. Our investigations concentrate on a complete design flow, analysis on high level Petri–Nets and their meaning for hardware/software partitioning of real-time embedded systems. The concepts for hybrid modeling of discrete and continuous systems are applied in an example in the domain of mechatronic systems.

Keywords

Design Method, Embedded Systems Design, Hardware/Software–Codesign, High Level Petri Nets

1 INTRODUCTION

In recent years embedded systems have gained increasing importance. This development is rooted in the growing number of systems containing embedded systems. Simultaneously the focus in design has shifted towards criteria requiring a common design process of the whole system. While classical hardware/software–codesign just considered optimizing the contrary aims of maximizing speed and minimizing costs, thinking about the fulfillment of hard real–time restrictions, minimization of the absolute size and weight of the assembly group, reduction of energy–consumption, and

*Supported by DFG–Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen" especially safety and reliability are getting more important now. 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. The individual components are typically partitioned into hardware components (ASICs, FPGAs) and software components (parallel real–time software on one or several general purpose processors) in order to ensure a realization fulfilling all requirements.

When modeling embedded systems it is important to handle concurrency. As shown in (Dittrich 1994) 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). Our work differs from their approach as we use Petri–Nets that support the notion of time and have a more powerful hierarchy semantic. We additionally consider verification and partitioning methods using the known methods for formal analysis of Petri–Nets.

Often embedded systems do not only contain discrete but also continuous system parts. In order to model those parts with a continuous behavior we transform them into a form that can be denotated by the event–based Petri–Net semantics. This transformation is basically a discretization described in (Brielmann 1995). In contrast to other methods that contain continuous time control, e.g., (Grimm *et al.* 1996), this allows us to analyze properties for hardware/software–partitioning purposes in one common model. Nevertheless a discretization means a loss of information in the model. For our purposes this disadvantage is not relevant as analysis or optimizations that need the original continuous informations (e.g. differential equations) are done on a higher level and an earlier design step. Using a combined graph–based model for analog and digital system parts as described in (Grimm *et al.* 1996) has the disadvantage that the analyzing facilities are limited to structural properties of the graph.

In (Tanir *et al.* 1995) Pr/T–Nets are used for analysis purposes either, but do not serve as a common model. The final implementation of the system is not compiled from the Pr/T–Net model but from their common specification language DSL.

In (Esser 1996) object oriented Petri–Nets are used as a common model for hardware/software–codesign. For this type of nets it will be very hard to develop analysis methods. Actually it is possible to simulate object oriented Petri–Nets for design validation.

As an example for an embedded system with discrete and continuous system parts we use a decentralized traffic management system, based on communicating autonomous units. Within the continuous system parts we have to deal with controllers for certain properties of the vehicle. Discrete system parts are responsible for event–oriented decisions within each vehicle and for communication between the vehicles to organize crossroad management. When considering each subsystem in isolation, the following problems would arise:

Coupling. 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 this level many predictions about the temporal and functional behavior of each subsystem can be made. To validate the behavior of the whole system, the individual models have to be coupled. 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 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. The prerequisite for this kind of analysis is that all models are given in an uniform language.

Analysis techniques. For every new modeling language the corresponding formal analysis methods always have to be re-developed and re-implemented. They are mostly based on classical analysis methods, e.g., the well–known ones for Petri–Nets, and represent an extension or adaptation to the specific areas of application. In order to avoid the expenditure of deducing new analysis techniques and to reuse existing methods, it is apparently more effective to use a known modeling language for the analysis that offers the desired methods suitable for the tasks at hand. So, we decided to use a special form of high–level Petri–Nets in our approach.

Continuous Systems. A lot of methods that have evolved in the area of hardware/software–codesign only consider discrete parts when partitioning the system into hardware and software components. As we can see in our application example continuous parts, e.g., controllers of continuous behavior, play an important role, too. For the purpose of performance estimations or formal verification it is necessary to have a common model for the discrete and continuous system parts. Thereby it is sufficient to transform continuous parts into a discrete form on an implementation–like level. On this level temporal and structural properties of these components are still existent.

Global Optimization. A further problem in a separated design process of different subsystems is that each individual domain has proprietary methods of optimization, but no facility for a global optimization of the joint system 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 neverbe reached and therefore can be eliminated from the design.

In the following sections we will give a brief introduction to our common formal model and an overview of our proposed design flow. We then describe strategies for hardware/software partitioning and formal analysis methods. Our application example shows how to model a hybrid system using extended Pr/T–Nets and how to apply some analysis methods for verification purposes.

2 FUNDAMENTALS ON PR/T-NETS

Pr/T–Nets were first introduced by Genrich and Lautenbach in (Genrich *et al.* 1981). They are bipartite graphs consisting of *places* usually depicted as circles and *transitions* depicted as rectangles (s. Figure 1). In order to define the behavior of a system, the places may contain *tokens*. 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 constant or variable tuples and transitions may carry first order formulas over a set of constants and variables. 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–Net is formally defined.

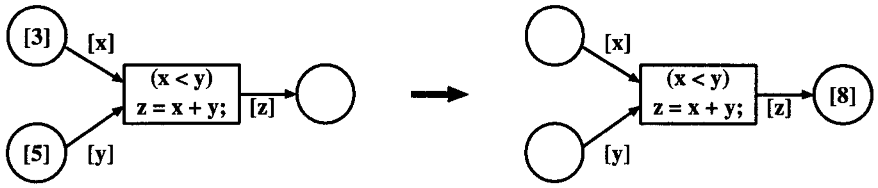

Figure 1 Pr/T–Net

Figure 1 shows a simple Pr/T–Net consisting of one transition with two input places and one output place. The tokens on the places are integers. The edges are annotated with tuples of variables ($[x]$, $[y]$, $[z]$) which determine the token flow over the edges. A condition is linked to the transition ($x < y$) which must be fulfilled by the tokens removed from the input places during the firing cycle of the transitions. Variables occurring on output edges of a transition ($[z]$) may be calculated using the *firing rule* of the transition ($z = x + y$). Transitions in an extended Predicate Transition Net are *enabled* (*can fire*) if appropriate tokens are available on all their input places. In this example the substitution of x by 3 and y by 5 determines a valid set of tokens for which the transition may fire, since $3 < 5$. The left side of Figure 1 shows the transition before firing and the right side after firing.

We extended Pr/T–Nets with several additional features. In this paper we will only explain the extensions which are necessary to understand the concepts for the design method for embedded systems. Information about the other extensions can be found in (Kleinjohann *et al.* 1996) or (Tacken 1992).

2.1 Extensions to Pr/T–Nets

Firstly, we extended the basic definition of Pr/T–Nets by a timing concept to allow the modeling of time dependent systems. In extended Pr/T–Nets an *enabling delay* and a *firing delay* (Starke 1990) can be defined for a transition. The enabling delay determines the time delay before a transition may become active after it has been enabled for any substitution and the firing delay specifies how long a transition in an extended Pr/T–Net 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.

Furthermore, extended Pr/T–Nets allow a hierarchical specification. Hierarchical specifications are useful to handle complexity in large designs and allow the reuse of predefined nets in several models. This is a necessity for the definition of libraries with subnets for special purposes. Furthermore, a hierarchical specification supports both a top down design as well as a bottom up design during the system specification. In extended Pr/T–Nets transitions and places can be refined by subnets. Such nodes 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 also have connections to nodes in the surrounding net. The places and transitions which are connected to nodes in the instantiating net are called port–places or port–transitions. The structured nodes are not simply replaced by their instantiated subnets. They have a special semantics which is defined via the activity of their subnet. A subnet of a structured transition is active as long as the structured transition itself is active which is similar to the philosophy of structured nets as described in (Cherkasova *et al.* 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 (Harel 1978).

2.2 Abstract graphical representation

Petri–Nets and also Pr/T–Nets have a standard graphical representation with places as circles, transitions as bars, and edges as arrows. For an engineer who has to decide whether a model works correctly this graphical representation is not easy to understand. Therefore, an abstract graphical representation which reflects the structure and state of a defined net would gain more acceptance.

Extended Pr/T–Nets provide the ability to define an intuitively understandable abstract graphical representation that is also capable to "continuously" represent the system's behavior and state changes during simulation. Hence, an engineer can define his own graphical representation of the system he is familiar with. This results in a more or less expressive animation. The description of the abstract representation may use arbitrary graphical elements. Hence, existing graphical specification languages can be produced by using their predefined symbols as the abstract representation of the Petri–Net elements. Furthermore, this graphical representation allows the user to interact with the system model in a natural way.

As an environment for a comfortable specification, simulation, and animation of extended Pr/T–Nets we use the SEA (System Engineering and Animation)–environment (Kleinjohann *et al.* 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*. In Figure 2 these design steps can be identified easily.

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 *parallel real–time–software*, *controllers having continuous behavior*, and *digital hardware*. This prepartitioning is based upon the experience of the engineer developing the system. 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. Already at this level a simulation can be performed by means of the corresponding, approved domain–specific tool. Eventually one may try to couple the simulations of several components and review them as a whole. There are two possible ways to transform models given in different specification languages (e.g. C++, SDL, Differential Equations, VHDL, Statecharts) into a Pr/T–Net model:

- *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). The existence of a transformation for the widely used specification language statecharts (Harel 1978) into Pr/T–Nets was proven in (Suffrian 1990). We are currently working on a transformation of SDL (Specification and Description Language). As such a transformation has to be implemented for each individual specification language we prefer another method for the SEA–environment described in the following.
- *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 tested this library method for the specification of data flow, block dia-

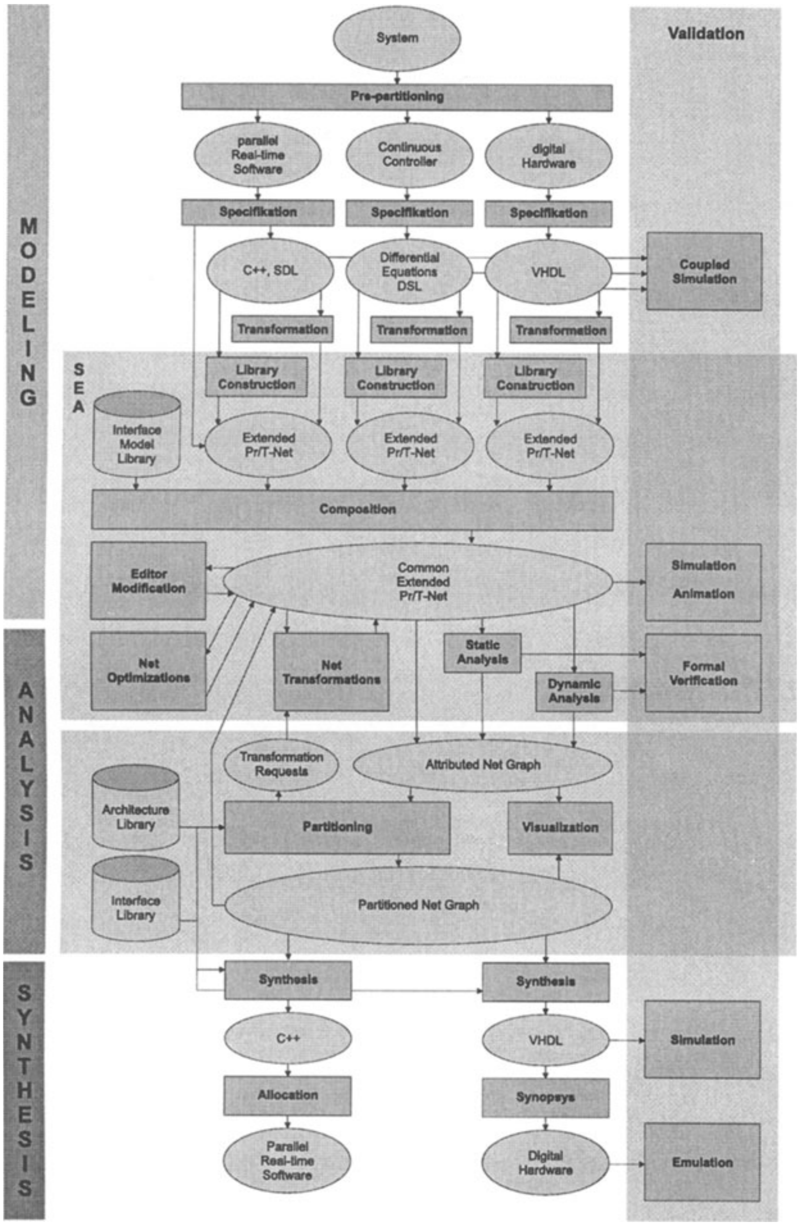

Figure 2 Design Flow

grams for differential equations, and asynchronous hardware on gate level, but other graphical languages are possible, 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 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 (s. Section 2.2).

After the process of modeling has been finished the design continues with the analysis and synthesis phase. In the following we describe strategies for those phases which are currently subject of our investigations. It is our aim to develop tools that support the presented methods for analysis and synthesis of our common specification language.

3.2 Analysis

In the analysis phase we have to distinguish between different purposes of the analysis. We consider analysis for the hardware/software partitioning and synthesis and for formal verification of certain system properties described in Sections 4 and 5.2. There are static and dynamic methods of analysis. The static methods are based on structural properties of the Petri–Net graph. When we talk about dynamic methods we think about the classical analysis methods for Petri–Nets that have to be expanded to the scope of extended Pr/T–Nets including occurrence, liveness, and safeness. Furthermore, timing annotations of the Pr/T–Net are considered for analysis. The results of this analysis can either be used for the partitioning or for the synthesis phase. During the partitioning phase some system parts are identified to be realized in hardware. The net representation of those system parts has to be transformed in order to ensure a hardware realization. An example for a net property of a hardware realization is the safeness of a place. A transformation reshapes the net in a way that the corresponding subnet does not contain any potentially overflowing places (ref. Section 4.2 c). To support an iterative design process, the whole net is re-exported into the modeling environment.

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 the specification languages of the known synthesis tools 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 VHDL–code that is synthesized by the SYNOPSYS–tools. Also on this level it is feasible to simulate the specification, either by executing the VHDL–code directly with a VHDL–simulator, or by emulating the synthesized FPGA–description by means of a hardware emulator. In the other case of parallel real–time software one compiles the corresponding Pr/T–subnet into parallel C++–code. In a first step we are trying to generate a kind of Pr/T–Net simulator in a compilation process. Another possibility is to identify certain net patterns by superimposing, matching, and replacing them with a known realization in C++. The resulting code will be much more lean compared to the one that includes a complete simulator.

4 PARTITIONING

The purpose of the partitioning phase is to divide the common Pr/T–Net specification w.r.t. the realization technique. That means that the structure of the partitions as well as the classification into the provided realization techniques digital hardware and parallel real–time software have to be determined. In this chapter we will describe how structural and behavioral analysis methods can be used for a hardware/software partitioning.

4.1 Level of Partitioning

Essentially for the usability of the results obtained by the partitioning is the choice of a suitable level of abstraction for the execution of the partitioning. A partitioning on system level divides the system into parts with specific functions. The advantage of functional partitioning is, that functionally depending blocks are not split in the synthesis phase. Mapping functionally depending blocks into different realization techniques results in a system hard to understand and hard to maintain. Depending on the granularity of the blocks a possibly more cost effective usage of resources is prevented, because parts of a functional block cannot be shifted to processors with a low utilization. Partitioning on a behavioral level allows a free choice of the boundaries of a partition. The realization technique is only determined by means of the behavioral analysis.

With regard to the Pr/T–Net modeling this means the following: During the modeling phase the designer defines possible partitions using hierarchy. This is already a more or less rough functional partitioning. However, this is also a restriction of the design space, which can only be avoided by transforming the entire hierarchical net into an equivalent flat net. Within this net partitions can be found on a behavioral level exceeding the boundaries of the original, hierarchical subnet. However, the experiences of the designer would be rejected when transforming a hierarchical net into a flat net. Thus it is suggestive to reduce hierarchical levels stepwise, i.e., in each step only one level of the hierarchy should be eliminated. The partitioning will be done on the flat net of the lowest level. This will be iterated as long as the results of the partitioning can be improved.

4.2 Analysis

At the level of the common specification language which is very far from a concrete realization it is very difficult to make realistic estimations of the performance and other properties of the system. The only chance to do hardware/software partitioning at this level is to use good heuristics which are applicable to our Pr/T–Net representation. In a first step we decided to concentrate on structural and timing aspects of the specification. Figure 3 shows a concrete strategy for analysis on the common Pr/T–Net. At first, the size and position of the partitions is determined as a result of a structural analysis of the net. It is suggestive to perform this step at the very beginning to reduce the complexity of the behavioral analysis. Therefore it has to be considered, that every edge in the Pr/T–Net crossing the boundaries of a partition is a communication link producing costs for the realization. The partitions have to be chosen in a way, that the number of communication links crossing the boundaries is minimal. This is a classical graph partitioning problem, which can be solved by using known methods, such as described in (Gajski *et al.* 1992). We are currently applying those methods using graph partitioning algorithms of the PARTY–library (Preis *et al.* 1996). Therefore we have to convert the Pr/T–Net into a graph using transitions as the only vertices and communication links as edges. The weights of transitions and communication links have to be determined w.r.t their annotations (number of operations in transitions and size of variables in communication links). The size of the partitions is given by the specification of the architecture which is taken from the architecture library.

The following two steps are based on the assumption that the intensive use of dynamics is not suitable for a hardware realization. For the generated partitions the use of recursion can be examined by means of the hierarchy graph. Those partitions are not suitable for a hardware realization because of the potentially infinite process dynamics. However, dynamic processes are not only produced by the use of recursion, but also by the use of the hierarchy semantics for structured places (s. Section 2.1). Such a dynamics can be recognized by means of a place invariance analysis of all places instantiating the same subnet.

In the next step timing annotations defining real–time restrictions are examined. For embedded real time systems it is not important to maximize speed as much as possible but it is necessary to guarantee that all subsystems always meet their deadlines. Depending on the dimension of the deadline annotations a realization technique can be assigned immediately. For very small deadline annotations it is necessary to synthesize special hardware, for very large or no deadline annotations a software realization suffices. In all other cases the runtime of the subnet has to be estimated for all possible realization techniques. Therefore the runtime of the preconditions and instructions in the transitions have to be evaluated step by step. We are currently adapting runtime estimation methods for C–code specifications used in (Hardt *et al.* 1995, Hardt 1995) to our Pr/T–Net based specification. If a software realization is preferred after this analysis, the real–time restrictions have to be classified. The operating system on which the software will be implemented has to be configured regarding the kind of real–time restrictions (hard, smooth, or no real–time restrictions). The fundamentals for this configuration are described in (Altenbernd 1995*b*, Altenbernd 1996, Altenbernd 1995*a*). In case of an identified hardware solution the corresponding subnet has to be transformed into a safe net to guarantee the existence of a hardware solution. In order to have no possibly overflowing buffers in the kernel of the net, potentially infinitely often markable places are moved to the boundaries of the subnet.

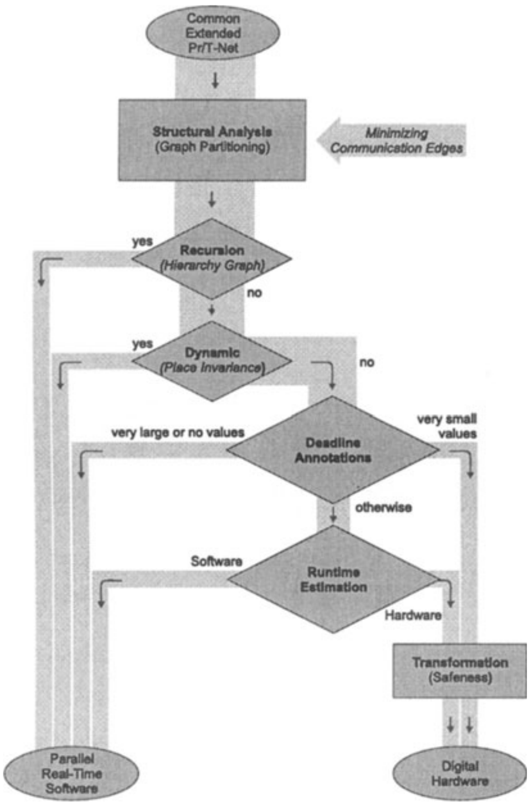
Common Extended Pr/T-Net
Structural Analysis (Graph Partitioning)
Minimizing Communication Edges
yes
Recursion (Hierarchy Graph)
no
yes
Dynamic (Place Invariance)
no
very large or no values
Deadline Annotations
very small values
otherwise
Software
Runtime Estimation
Hardware
Transformation (Safeness)
Parallel Real-Time Software
Digital Hardware

Figure 3 Partitioning

(a) Detection of recursion

In extended Pr/T–Nets recursive Nets can be specified using the instantiation mechanism. A net can be used as a subnet of a structured node which is defined in the net itself. This is possible because the structured node only contains a reference to the subnet and the dynamic information is copied only if necessary (a structured place is marked or a structured transition becomes active). An extended Pr/T–Net cannot be realized as hardware if it contains recursion. Therefore it is necessary to detect such recursive definitions in a specification.

To show which nets are used as subnets in other nets a hierarchy tree can be built. The nodes in this tree are the used subnets and the edges denote the instantiation dependencies. But if the hierarchical definitions contain recursive specifications this hierarchy tree becomes infinite. Figure 4 a) shows such an infinite tree. The net N_1 instantiates the net N_2 two times and the net N_3 once. The net N_2 instantiates N_4 which instantiates N_3 which again instantiates N_2 and so on.

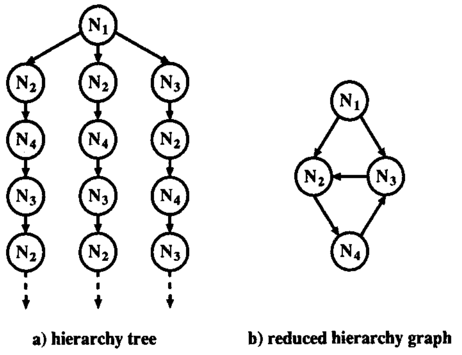

Figure 4 Example hierarchy tree and corresponding graph

To avoid such an infinite tree a reduced hierarchy graph can be built. In this graph each net occurs only once as a node. Figure 4 b) shows the corresponding reduced hierarchy graph for the hierarchy tree in Figure 4 a). If the reduced hierarchy graph is built the detection of recursion is reduced to the detection of cycles in the graph.

(b) Dynamics

The main purpose of this analysis is to detect if a subnet which is used several times in a specification may also be active several times or perhaps only once. For example a subnet which calculates a complex computation is used several times to refine structured places. This subnet should be realized as a functional hardware block and it has to be determined, if one or several hardware components are needed. The subnet of a structured place is only active if the structured place is marked. To decide whether such a subnet may be active several times one needs to check if more than one of the structured places is marked at once. This can be done with a place invariance analysis as described in (Genrich *et al.* 1981) and (Genrich 1987). If a set of places is recognized as an invariant this means that the number of tokens on these places is constant for every firing sequence of the net.

(c) Transformations

If a subnet is chosen for a hardware realization, one has to assure that the subnet is *k–safe*. K–safeness means that no place in the subnet contains more than *k* tokens for every possible firing sequence. If some places do not fulfill the k–safeness criterion, the subnet has to be transformed by moving the unsafe places to the boundaries of the subnet. This transformation is an equivalence transformation, thus the behavior of the subnet does not change and only buffer functionality is moved to the boundaries. The k–safe kernel of the subnet can then be implemented in hardware, while the border, i.e., the interface of the subnet to the environment, has to be implemented as software. The required transformations for Pr/T–Nets can be deduced from transformations suggested in (Kleinjohann 1994). There transformations for the implementation of Pr/T–Nets as asynchronous hardware are described.

5 APPLICATION EXAMPLE

The concepts of hybrid modeling based upon a common Pr/T–Net model have been applied to an example from the area of mechatronics, in our case a decentralized traffic management system. The idea of this system is to equip vehicles with an intelligent control allowing an optimal run of the vehicles under certain criteria by means of communication between the vehicles. Such criteria are avoidance of collision, energy consumption, pollution emission, noise reduction, and flow-rate. In order to increase the flow–rate of vehicles on a road it is suggestive to build motorcades.

5.1 Modeling

Figure 5 shows the structure of a vehicle with the ability to build motorcades autonomously. It is assumed that all vehicles are connected via a communication channel. By means of a special protocol those vehicles eligible for building a motorcade are found and an optimal velocity and other parameters are negotiated. Beside this discrete part of information processing there is a continuous part controlling the velocity and distance to the vehicle driving ahead. This controlling is based upon a measurement of distance and velocity by means of corresponding sensors.

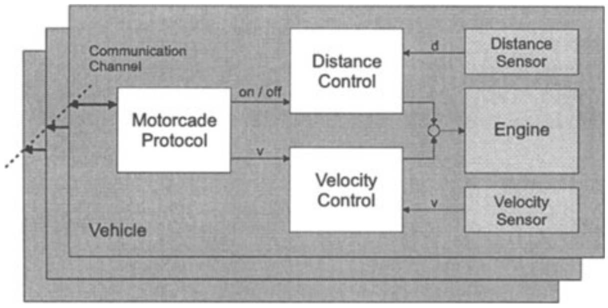

Figure 5 Global Structure of a Vehicle

In Figure 6 the protocol for building a motorcade is shown as a SDL-specification.

Initially the vehicle is in the state *Searching* sending a search signal (*MSearchReq*) periodically to all other vehicles. Once a vehicle answers the state is switched to *Building*. If the distance and direction of the vehicle allows the building of a motorcade the leading vehicle and the optimal velocity is determined. Otherwise the building of the motorcade is refused (*MRefuseReq*). If the own vehicle is the leader of the motorcade the computed velocity is transferred to the velocity control and the state changes to *Leading*. Otherwise the distance control is switched on and the state becomes *Following*.

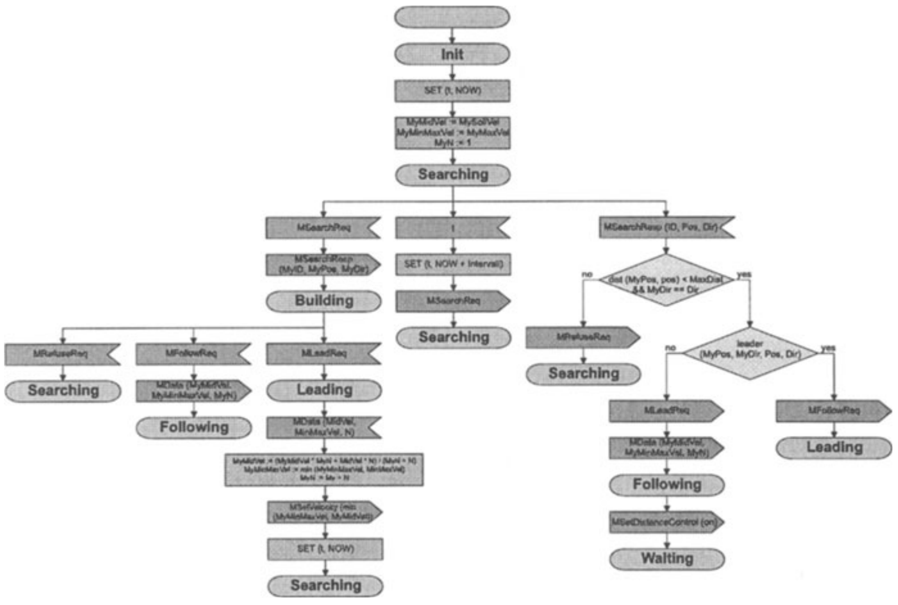

Figure 6 Motorcade Protocol in SDL

This modeling can easily be transformed into a Pr/T–Net. Therefore a place is modeled for each state of the SDL-diagram. There are two additional places, one for receiving input signals and one containing output signals to be sent. A state transition is represented by a transition in the Pr/T–Net. The preconditions of this transition are the place with the input signals and the place representing the corresponding state. The predicate of the transition formulates a condition only allowing tokens representing the corresponding input signals. The transition then generates a token representing the output signal. This token is fired into the output place while a control token is fired into a place representing the following state. Additional places for the variables used in the model have to be specified. Every transition where such a variable is needed has to be connected to the corresponding place. This is partly performed in Figure 7.

In Figure 8 the distance control is shown as a block diagram. The controlling is done by leading the measured distance signal back to the input of the controller.

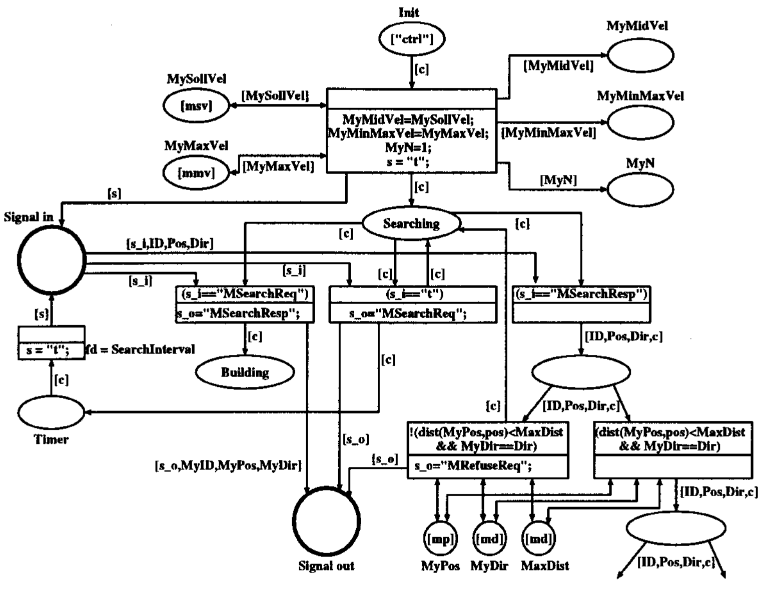

Figure 7 Motorcade Protocol as a Pr/T–Net

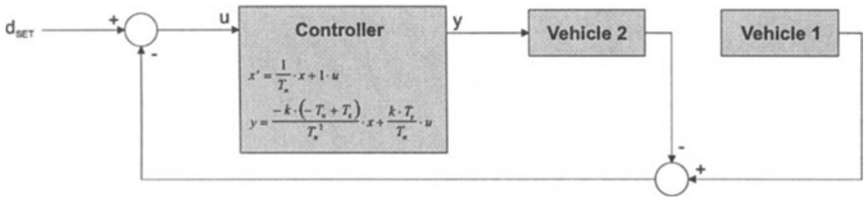
d_SET
u
Controller
$x' = -\frac{1}{T_e} \cdot x + 1 \cdot u$
$y = \frac{-k \cdot (-T_e + T_i)}{T_e^2} \cdot x + \frac{k \cdot T_i}{T_e} \cdot u$
y
Vehicle 2
Vehicle 1

Figure 8 Block Diagram of the Distance Control

The controller itself can be described by a differential equation, in this case already given in its state form. From this form a discretized representation can be found which can be transformed into a Pr/T–Net (s. Figure 9.) The process of discretization is basically the implementation of a numerical integration method computing the integral by a stepwise summation. The step width *h* has to be determined w.r.t. the required precision. This transformation is described in (Brielmann 1995) in more detail.

5.2 Correctness

As mentioned before there are many classical methods for analysis on Petri–Nets. We applied those methods to prove the correctness and certain properties of our application. Examples for those properties are the safeness and deadlock freeness of our traffic management system, in this case the collision avoidance mechanism of

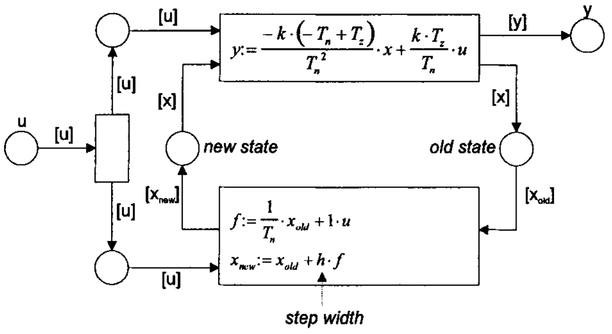
u
[u]
[u]
[u]
[u]
[u]
$y := \frac{-k\cdot(-T_n+T_z)}{T_n^2}\cdot x + \frac{k\cdot T_z}{T_n}\cdot u$
[y]
y
[x]
[x]
new state
old state
$[x_{new}]$
$[x_{old}]$
$f := \frac{1}{T_n}\cdot x_{old} + 1\cdot u$
$x_{new} := x_{old} + h\cdot f$
step width

Figure 9 Controller of the Distance Control as a Pr/T–Net

an intersection. We modeled such a mechanism by an extended Pr/T–Net based on critical regions in an intersection. Figure 10 a) shows the intersection with the critical regions A_1 to A_{12}. There should never be more than one car in such a critical region.

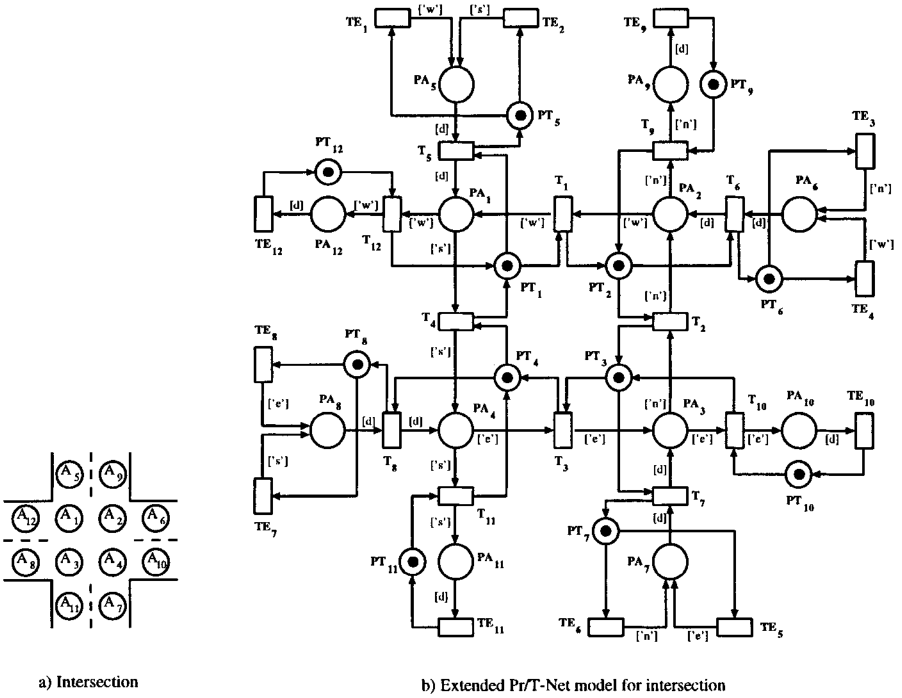

Figure 10 Intersection with extended Pr/T–Net model

Figure 10 b) shows the extended Pr/T–Net model. In this simple example the cars may only cross the intersection straight forward or turn right. The critical regions are modeled by places PA_1 to PA_{12}. Each of these places has a corresponding semaphore place PT_1 to PT_{12} that initially contains a simple token (a token with no special value). The cars are modeled by tokens with a value containing their desired direction ('n', 's', 'w' or 'e'). The tokens for the cars are produced by the transitions TE_1 to TE_8 moved by the transitions T_1 to T_{12} and at last consumed by the transitions TE_9 to TE_{12}. Every time a car enters a critical region the transition that models the entering needs the simple token from the corresponding semaphore place. The semaphore places guarantee a safe crossing of the intersection. This can formally be proven by linear analysis methods (building the so called invariance matrix and compute place invariants). The results are that all the pairs of places $\{PA_1, PT_1\}$ to $\{PA_{12}, PT_{12}\}$ are invariant, i.e. for every firing of any transition the number of tokens on these pairs of places is constant. The initial number of tokens on each pair is set to one, i.e. all these places are 1–safe and so the whole net is 1–safe. If a net is safe the state space (occurrence graph) is finite and can therefore be computed. Based on the state space more properties of the net like liveness or possible deadlocks may be proved.

Because of the variety of different possible markings for the net in Figure 10 b) the whole state space contains 104 976 states. We tried to compute this state space using the DesignCPN–Tool (Jensen *et al.* 1996) (after transforming the net into an equivalent colored net (Jensen 1992, Jensen 1994)) on a SPARC Ultra Station with 128 MB of main memory and 130 MB of swap space but failed. After 16 hours of computing the tool gave up because their was not enough memory to compute the whole design space. Therefore we reduced the net by omitting the critical regions A_5 to A_{12} as shown in Figure 11 a). This resulted in an occurrence graph with 81 states that could be computed by the DesignCPN–Tool. We found out that the defined net has a deadlock which occurs if there are four cars in the regions A_1 to A_4 that all want to drive straight forward. In this case the places PA_1 to PA_4 are marked with $['s']$ $['w']$ $['n']$ and $['e']$.

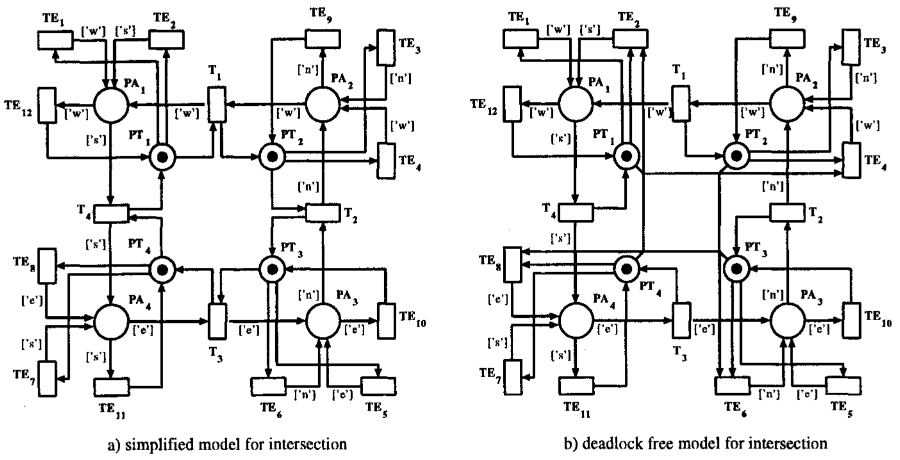

Figure 11 Reduced extended Pr/T–Net model with and without deadlock

To avoid this deadlock we redesigned the net model. If a car wants to go straight forward the corresponding transition has to allocate the tokens for both used critical regions in advance. The resulting net is depicted in Figure 11 b). For this net we could prove that it is deadlock free and that the liveness property holds for all its transitions.

6 SUMMARY AND OUTLOOK

Our suggested design flow is an approach towards an integrated, complete embedded system design. We have shown how to use an extended form of Pr/T–Nets as a common model for specification. Due to their well known analysis methods we think that Pr/T–Nets are suitable as a common modeling language. The application example shows that hybrid modeling of discrete and continuous systems is possible, although continuous systems have to be discretized. For our proposed target architecture this is no disadvantage because continuous controllers usually will be implemented in their discretized form (e.g., DSPs compute continuous behavior in a discrete data flow). We also applied the classical Petri–Net analysis methods for the purpose of formal verification. As formal verification is a difficult task there is another possibility to validate the system. The SEA–environment has a built–in simulator and animator for testing the common system specification. For an automatic hardware/software partitioning we investigated different analysis methods on extended Pr/T–Nets. Especially the use of dynamics, i.e., the use of recursion, process dynamics, and potentially unrestricted places, can be recognized using Pr/T–Net analysis methods. Our further investigations concentrate on optimizations using local transformations on the Pr/T–Nets.

REFERENCES

Altenbernd, Peter. 1995*a*. Allocation of Periodic Hard Real-Time Tasks. In *20th IFAC/IFIP Workshop on Real Time Programming (WRTP)*. Fort Lauderdale:

Altenbernd, Peter. 1995*b*. Deadline-Monotonic Software Scheduling for the Co-Synthesis of Parallel Hard Real-Time Systems. In *Proceedings of the European Design and Test Conference (ED&TC)*. Paris: .

Altenbernd, Peter. 1996. On the False Path Problem in Hard Real-Time Programs. In *8th Euromicro Workshop on Real Time Systems (WRTS)*. L'Aquilla: .

Brielmann, Maria. 1995. Modelling Differential Equations by Basic Information Technology Means. In *Proceedings of the 5th International Conference on Computer Aided Systems, Theory and Technology (EUROCAST'95)*. Innsbruck, Austria: .

Cherkasova, L. A. & V. E. Kotov. 1981. Structured Nets. In *Mathematical Foundations of Computer Science*, ed. J. Gruska & M. Chytil. Vol. 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*, ed. Jerzy Rozenblit & Klaus Buchenrieder. New York, NY: IEEE Press chapter 6.

Esser, Robert. 1996. An Object Oriented Petri Net Approach to Co-design. In *Design Automation for Embedded Systems*. Schloss Dagstuhl, Germany: .

Gajski, Daniel, Nikil Dutt, Allem Wu & Steve Lin. 1992. *High Level Synthesis — Introduction to Chip and System Design*. Kluwer Academic Publishers chapter Partitioning.

Genrich, H. J. 1987. Predicate/Transition Nets. In *Advances in Petri Nets 1986*, ed. W. Bauer, W. Reisig & G. Rozenberg. Vol. 254 Springer Verlag. Part I.

Genrich, H.J. & K. Lautenbach. 1981. "System Modelling with High-Level Petri Nets." *Theoretical Computer Science* 13.

Grimm, Christoph & Klaus Waldschmidt. 1996. KIR — A graph-based model for description of mixed analog/digital systems. In *Proc. of IEEE Euro-DAC/Euro-VHDL*. Genf, Switzerland: .

Hardt, Wolfram. 1995. An Automated Approach to HW/SW-Codesign. In *IEEE Colloquium: Partitioning in Hardware-Software Codesigns*. London, Great Britain: .

Hardt, Wolfram & Raul Camposano. 1995. Specification Analysis for HW/SW-Partitioning. In *3. GI/ITG Workshop: Anwendung formaler Methoden für den Hardware-Entwurf*. Passau: .

Harel, D. 1978. "Statecharts: A visual formalism for complex systems." *Science of Computer Programming* 8.

Jensen, K., S. Christensen, P. Huber & M. Holla. 1996. *Design/CPN. A reference manual*. University of Aarhus: Computer Science Department. Online: http://www.daimi.aau.dk/designCPN/.

Jensen, Kurt. 1992. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use*. Vol. 1, Basic Concepts of *EATCS Monographs on Theoretical Computer Science* Springer Verlag.

Jensen, Kurt. 1994. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use*. Vol. 2, Analysis Methods of *EATCS Monographs on Theoretical Computer Science* Springer Verlag.

Kleinjohann, B. 1994. Synthese von zeitinvarianten Hardware-Modulen PhD thesis Universität Gesamthochschule Paderborn.

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

Preis, R. & R. Diekmann. 1996. *The PARTY Partitioning – Library — User Guide — Version 1.1*.

Starke, P. H. 1990. *Analyse von Petri–Netz–Modellen*. Leitfäden und Monographien der Informatik Stuttgart: Teubner.

Suffrian, U. 1990. Vergleichende Untersuchungen von State–Charts und strukturierten Petri–Netzen. Master's thesis Universität Gesamthochschule Paderborn.

Tacken, J. 1992. Steuerung und Überwachung von Entwurfssystemen mit Hilfe von Prädikat/Transitions–Netzen. Master's thesis Universität Gesamthochschule Paderborn.

Tanir, Oryal, Vinod K. Agarwal & P.C.P. Bhatt. 1995. "A Specification-Driven Architectural Design Environment." *IEEE Computer* 6:26–35.

7 BIOGRAPHY

Bernd Kleinjohann studied computer science at the University of Dortmund and received his *Diplom* in 1985. Since then he has been with C-LAB, formerly Cadlab, a cooperation between University of Paderborn and Siemens Nixdorf Informationssysteme AG. In 1994 he received his PhD from Paderborn University. His current research focusses on Petri Nets, engineering of real time systems, especially of asynchronous systems and embedded systems.

Jürgen Tacken received his *Diplom* in computer science from the Paderborn University in 1993. He is currently researcher at C–LAB and works towards his PhD. His main interests are the specification and verification of heterogenous systems using Predicate/Transition–Nets.

Christoph Tahedl did his diploma thesis at Paderborn University in 1995 in the field of visual languages. He is now PhD student at Paderborn University and works on partitioning in system specifications and hardware/software–codesign of embedded real time systems.