Modelling Aspects of System Level Design

Franz J. Rammig
Universität-GH Paderborn/FB 17
33098 Paderborn
Germany
Tel.: (+49)5251-602069
Fax: (+49)5251-603427
e-mail: franz@uni-paderborn.de

June 17, 1993

Abstract

In this paper the neccessity for a common modelling approach for heterogeneous systems is discussed. As an example for such a modelling technique extended Predicate/Transition Nets are introduced. These nets can combine modelling in a declarative way by means of first order logic with an operational interpretation inherited from Petri Nets. The added concepts of hierarchy and recursion allow the description of extremely complex systems. Three applications of Pr/T-Nets are shown: Control of complex design systems, timing analysis and the implementation of communication protocols.

1 Introduction

A basic problem when designing complex heterogeneous systems is to find a uniform method to model the entire system, i.e. all the different aspects such a system has. Only when such a model exists there is a chance to provide automated tools (qualitative and quantitative ones) for design activities like system partitioning and decision support concerning the target technology to be used for the different parts of the system. In this paper it will be argued that extended Predicate/Transition nets form an adequate modelling environment for this purpose. The main reason why Pr/T-nets are so attractive is their flexibility. While axiomatic approaches have problems to model operational and timing aspects in an adequate manner (especially in the case of concurrency) most operational models fail to support easily rigorous design methods like term rewriting and formal proofs of correctness. The latter is especially true for most software engineering approaches. But also operational approaches with a precise semantical foundation like Statecharts have the deficiency that various aspects of a system are modelled either by different diagrams or in a mixture of explicit and implicit notations. With Pr/T nets in one single document the data flow and the control flow are specified in a consistent way. The modelling style can be tuned towards a pure predicate logic specification in one extreme, to a pure operational description in the other one, or to any mixture of both with slight transitions. This flexibility also allows to apply various classes of algorithms on such models. Graph-oriented methods as being followed up in the context of timing analysis, proof techniques that operate on a predicate representation, synthesis algorithms (especially scheduling and allocation working on dataflow and control flow representations) that need operational information, they all can extract the information relevant for them from one single modelling framework.

2 Extended Predicate / Transition Nets

2.1 Basic Pr/T-Nets

Predicate/Transition Nets (Pr/T-Nets) initially have been introduced by Genrich and Lautenbach [10] in 1981. What makes them so attractive for system modelling is the fact that both, declarative concepts (first order logic) and operational ones (Petri Nets) are present in this model. In addition a representation of data flow and control flow in one single document is provided, what in contrast to notations like Statecharts makes closed representations in one sin-

gle document possible. Pr/T-Nets have an easy to understand graphical notation. On the other hand they have the computational power of Turing Machines [10]. We here extend the original model by the representation of hierarchy and recursion. With these extensions all classical techniques to attack complexity are present: decomposition (divide and conquer), hierarcharchy and recursion. Mathematically Pr/T-Nets form relatively complicated mathematical objects. Therefore only a short informal introduction is given in this paper. Like in ordinary Petri Nets the underlying graph theoretical structure is a bipartite graph with two classes of nodes: transitions to model actions and places to model conditions. As in ordinary Petri Nets places may be marked with tokens and there is a token game played locally by transitions due to the marking of the input places of the individual transitions. In contrast to ordinary Petri Nets in Pr/T-Nets tokens are individuals. In Pr/T-Nets a token is an instantiated object of a certain type. Assuming an underlying Petri Net Graph $PG = (P,T,F,B)$, P finite set of places, T finite set of transitions,
$P \cap T = \emptyset$,
$F \subset \{(t,p) | \ t \in T, p \in P\}$,
$B \subset \{(p,t) | \ p \in P, t \in T\}$,
a global marking M now is a bijective mapping from the set of places P onto the set of instantiated tokens,
$M : P \rightarrow \{Ko_i \in \mathbf{P}(K_o) | K_o =$
set of instantiated tokens,
$K_{oi} \cap K_{oj} = \emptyset$ if $i \neq j, \cup K_{oi} = K_o\}$.
I.e. tokens are instantiated only as part of the marking of a place and each instantiated token marks only one single place. Input arcs of a transition $t \in T$ are labelled with typed variables. A transition is firable only if there is a valid interpretation of the set of these typed variables using currently instantiated tokens in the respective places. Equally named variables attached to input arcs of one transition have to be substituted by the same values for an interpretation to be valid. Transitions have attached a predicate and a token mapping. A transition t fires under a specific interpretation only if this predicate is true for this interpretation. So looking for a valid interpretation may be interpreted as looking for sufficient syntactically correct data while testing the predicate means testing whether semantical restrictions are met. In the case of more than one valid interpretation which result in the predicate to become true, the transition fires independently for these interpretations. If a transition fires, it destroys the input tokens that constitute the considered interpretation and calculates (i.e. instantiates) tokens on its output places. By labelling output arcs with typed variables, values can be routed individually to token instantiatons. The values to be assigned are calculated due to expressions that can be attached arbitrarily to transitions.

Example 1 :

Fig. 1 shows a single transition before firing, Fig. 2 after. If we assume that the type of all variables is *integer* there are two valid interpretations: $(x \rightarrow 4, y \rightarrow 5)$ and $(x \rightarrow 8, y \rightarrow 7)$. Only the first one is accepted by t for firing, as the transition's predicate requires $x < y$.

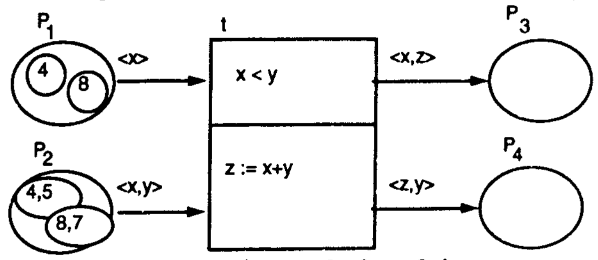

Fig. 1 Pr/T-net before firing

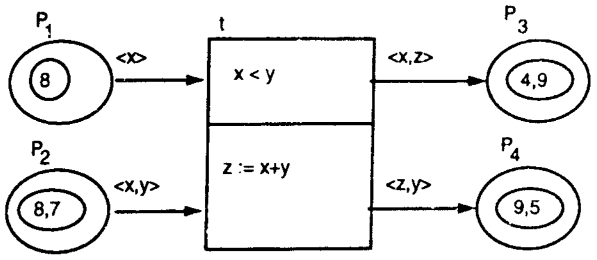

Fig.2 Pr/T-net after firing

In this context we will concentrate on two additional extensions: hierarchy and recurion. The first one combines a technique introduced by L. Cherkasova [CK81] for ordinary nets with the concepts of Pr/T-Nets. This approach has first been reported by Kirstein [14]. Concerning recursion we refer to an own result published in [15].

2.2 Hierarchical Pr/T-Nets

The problem when introducing hierarchy to any kind of Petri Nets is to handle structure and behaviour in a consistent manner. Obviously there is no problem to define a hierarchical structure in an isolated manner; each schematic entry system is capable to do this. We use the same technique to define hierarchical structures of Pr/T-Nets here. The underlying structural concept of Pr/T-Nets is the Petri Net graph $PG = (P,T,F,B)$. To avoid complicated graph grammer approaches, a simple port/port-map technique similar to VHDL is used here. Each transition's or place's input/output edge may serve as a port, where each port may be of one of three types *in*, *out*, *inout*. The set

of all ports of a net forms its interface. Graphically a place-port is drawn as a solid dot, a transition-port as a solid rectangle and the port-type is denoted by an arc between the port-symbol and the node being a port. As in Pr/T-nets the individual edges of a transition t have individual meanings, they are indentified by $t.a$, where a is the edge's attribute. Hierarchy is now introduced by allowing a transition of a (hierarchical) Petri Net graph to instantiate a (hierarchical) Petri Net graph with interface. By this process a copy of the graph to be instantiated is made and located internally of the instantiating transition. After this copying process the ports have to be connected. For this purpose formal ports are introduced at the instantiating transition. They too may be of kind *place* or *transition*. From the outside these formal ports may be connected to places or transtions due to the rules of well formed Petri Net graphs, i.e. a formal port of kind transition may be connected to a place and vice versa. Internally the interfaces of the instantiated net have to be mapped onto the formal ports of the instantiating transition. Obviously a one to one correspondance between formal ports and actual ones is required.
Concerning the behaviour, the basic approach of [6] is applied to Pr/T-nets. In the following an instantiating transition, i. e. a transition with embedded net is also called a *macro transition*. For such a macro transition the following behaviour is defined: It becomes firable by the same condition as a normal one and plays the first part of the token game up to the destruction of the used input tokens like a normal one. Internally the firing of a macro transition means that the (always identical) initial marking is taken. Starting from this marking the embedded net becomes active and remains active as long as it is live. When the embedded net becomes dead the macro transition plays its remaining token game in the same way a usual Pr/T-net transition does. At the same time the internal initial marking is restored. As there may exist connections from an embedded net to external places, a fine grain synchronization with external behaviours is possible. This definition of a hierarchical Pr/T-net behaviour is a very natural one: When an expandable transition is activated it starts its embedded task and remains active as long as the embedded net (i.e. the task to be performed) has not been finished. After this the macro transition reports to the outside that it has performed its task and prepares for reactivation. It can be seen immediately that this behavioral hierarchy is completely consistent with the structural one.

2.3 Recursive Hierarchical Pr/T-Nets

Up to now no restriction has been introduced that forbids a macro transition to instantiate a net that contains this macro transition. Structurally such an instantiation means recursion. Being structurally no problem the exact semantical meaning of such a constellation has to be defined. For this purpose the technique of *coloured Petri Nets* is used, a technique which is already included in the framework of Pr/T-nets, as each token-type may include a colouring component. Whenever there is a recursive net definition it is implicitely assumed that the tokens and the firing rules are coloured ones. For convenience in the sequel we will use the term *recursion-path* instead of colour. For this purpuse the recursively defined macro transition and aech of its occurrencies inside the recursive definition have to have a unique identification. A concatenation of such identifications serves as a unique recursion path, identifying for each token to which recursive activation it belongs. Initially the recursion-path is set to the identification of the entire macro-transition. When a subnet has to be initiated recursively, its input tokens are copied but with a recursion-path attached, which is the caller's recursion-path with the identification of the called occurence in the definition concatenated. If a recursively activated macro-transition is terminated, it marks its output-places (as usual) but with a recursion-path attached which is the actual internal recursion-path with the last identification symbol truncated. By this technique obviously just the usual techniques to handle recursion have been expressed within the framework of hierarchical coloured nets. A recursive definition, however, would be useless when not ancered. Therefore a macro-transition that is refined in a recursive way (obviously by definition recursion can occur only for macro-transitions) has to have two exclusive paths connected to the same external input/output places and the predicates of the two pathes have to be mutually invers. One path, serving as the recursion's ancer must not be refined recursively.
Example 2:
The recursive Pr/T-net as shown in Fig. 3 performs *Quicksort*. The macro-transition *qs* takes a list of integers as input an produces a sorted list of these integers. The embedded transition *ancer* serves as ancer of the recursive definition: if the input-list is empty an empty output list is produced. In all other cases first the transition *hd* extracts the first element (x) of the input list and hands over the remaining list (xs) together with x to transition *part*. This transition partitions xs to a sublist (S) of elements that are smaller

than x and another one (L) of such elements that are larger than x. Both sublists now are sorted by recursively activating *qs*. *L1* is the sorted version of list L, *S1* the sorted version of list S. Transition *concat* forms the list z by concatenating x left to *L1* while transition *appnd* concatenates the sorted sublists (now including x). The entire control-flow is synchronized by places local to *qs* (note that by definition places connected to formal ports are shared ones) in such a way that there is a sequence: *hd* then *part* then two recursive activations of *qs* concurrently then (after termination of *concat* wich is activated after the termination of the *qs* instance for sublist *L)* *appnd*. The dynamic behaviour is easy to see when trying to play the coloured token game with a small example.

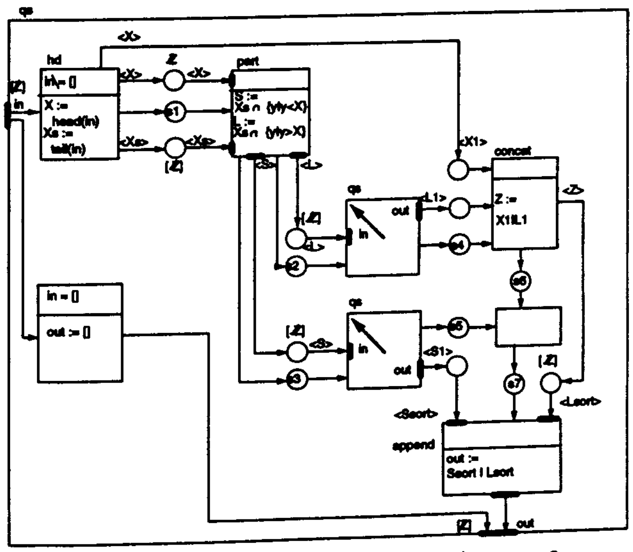

Fig. 3: Hierarchical recursive Pr/T-net for quicksort

2.4 Modelling with Hierarchical Pr/T-Nets

The above discussion has shown that hierarchical recursive Pr/T-nets form a very powerful means to model complex systems in a convenient way. Readability of this graphical language is provided mainly by the very simple principles of Pr/T-nets which inherited simplicity and locality from original Petri Nets but add the expressive power to specify both, control and data-operations. With the features hierarchy and recursion added (as described above) all basic techniques to conquer compexity are present. A complex system to be specified now can be decomposed into its basic components where each component is modelled as a macro-transition. By identifying the types of objects that such a macro- transition consumes and produces, the basic information streams are clearly defined. Finally the concurrent overall control structure can be expressed using the usual Petri Net principles. The hierarchy concept now allows the macro-transitions to be refined. This may be carried out in a stepwise manner until an intended level of abstraction has been reached. Where adequate the refinement can be given in a recursive way. The refinement process may diverge into different implementation techniques: software where a scheduler takes over the role of the general control structure [16], hardware where the control structure may be mapped directly into an implementation, or any mixture (HW/SW-codesign [9]). There are rarely techniques published to perform the partitioning process into hardware and software components automatically. In principle feature-oriented qualitative methods are applicable or quantitative ones. In both cases hierarchical Pr/T-nets form a proper base for such algorithms. Features like "high degree of recursion" that make a software solution for a part of a specification advisable or "high degree of concurrency" that imply preference of a hardware implementation of the respective part can be identified easily. Quantitative methods usually start from the fiction of a pure software implementation [9] or a pure hardware solution and stepwise replace parts of such a pure implementation by the opposite technique until certain restrictions are met. For such an approach a metric (or a couple of metrices) is neccessary. Concerning this metric the model can be profiled, e.g. by observing various simulation runs. On the other hand, based on the same metric, an objective function can be defined that governs the optimization process. This process itself then can be carried out by any generic algorithm like Simulated Annealing, Genetic Algorithms, Simulated Neural Networks, Linear Programming, etc. What is important in our context is that the neccessary metrics can be included easily in specifications given by recursive Pr/T-nets without extending the model.

3 Hierarchical Pr/T-Nets Applied to Controlling Complex Design Systems

One of the most complex systems to be designed are design systems. As complex design problems can be solved in acceptable time only by a a group of designers working concurrently, support is neccessary to control a well organized cooperation of such a team. As this problem has been identified by various practitioneers and researchers, design flow managers became part of design frameworks and a couple of scientific papers on this topic have been published. It is interesting that a relative high number of these papers use net-based models for their approaches [4, 7, 8, 13]. Here a short excerpt from the approach of Lisa Kupitz and Jürgen Tacken [13] shall be given as these authors explicitely make use of the power of hierarchical Pr/T-nets. Their design monitor system is called DECOR (Design Control and Observation) and is tightly integrated into the JESSI Common Framework [17] using the same integration techniques used for the tools to be monitored themselves. DECOR provides generic design monitoring and control facilities that allow online monitoring of design actions down to the granularity of elementary object manipulations. In DECOR design tools typically are modelled as a hierarchical place that contains a subnet that models the tool's individual actions. Tokens that mark such a hierarchical place represent the processes that currently use this tool while tokens marking places inside this hierarchical place represent individual states of such a process' execution. DECOR consists of a net-editor that allows to create hierarchical Pr/T-nets of arbitrary complexity, a global execution monitor that executes such a net and by this controls the usage of a design environment, and for each user optionally a dedicated monitor. An event/trigger mechanism allows a fine granular cooperation between DECOR's components and to monitor tool actions down to singular object access. E.g. "drilling" a hole into the cabinet of an electronic system with a mechanics editor may cause a specific transition of the controlling Pr/T-net to fire and by this an EMC-tool may be activated that calculates the resulting change of radiation. This example illustrates DECOR's excellent suitability to serve as controller in a computer aided concurrent engineering environment.

4 Hierarchical Pr/T-Nets Applied to Timing Analysis

As mentioned above a basic prerequisite for any kind of quantitative decomposition in HW/SW-codesign is the availability of metrices. Among these metrices timing information plays an important role. In fact it is timing that mostly prevent designers to restrict themselves to pure software solutions. In [1] P. Altenbernd and R. Milczewski desribe an elegant way to carry out timing analysis within the framework of Pr/T-nets. In fact in their paper they even restrict themselves on ordinary Petri nets which form a subclass of Pr/T- nets. Their approach is not only applicable to electronic circuits at the gate level but also to high level models and heterogeneous mechatronic systems. Their basic idea is to model not only the system to be analyzed by a Pr/T-net but also the constraints to be checked for. E.g. in their paper they present net templates, that check for missing events or for forbidden ones. The model of the system to be checked and the constraints then are combined automatically into a homogeneous model that either can be analyzed in an hierarchical way using Cadlab's timing analysis system CaTAS [2] or may be simulated by any Pr/T-net simulator.

5 Hierarchical Pr/T-Nets Applied to Communication Systems

Signal Transition Graphs (STG) [5] are a well known technique to model asynchonous systems and to synthesize asynchronous hardware from such models. They form a rather restricted class of Petri nets and by this an even more restricted class of Pr/T-nets. In [12] B. Kleinjohann and R. Milczewski show how this synthesis appraoch can be extended to more general hierarchical Pr/T-nets. On the other hand in [3] M. Brielmann and B. Kleinjohann show how even time continuous systems may be modelled by means of Pr/T- nets. As conventional synchronous systems also are covered by this model all conventional design techniques are available as well. What makes Pr/T-nets so attractive for system design, however, is the fact that software implementations may also be derived very easily. In [16] M. Rupprecht describes a method to specify complex communication protocols by means of Pr/T-nets and to synthesize automatically a software realization from such a specification. Conceptually the underlying abstract machine architecture is what he calls a Multiple Decision Multiple Action (MDMA) machine. In such an architecture "action units" execute threads of sequential "protocol modules" (POM). They do this under the control of a "decision unit" which executes the controlling Pr/T-net. Such a MDMA-machine may be built as dedicated protocol hardware. Another alternative described in [16] is to program a conventional v.Neumann processor using the programming language PENCIL/C. This language has been developed and implemented as dedicated language to describe communication protocols based on Pr/T- nets. The basic component of the software implementation of PENCIL/C is a scheduler that plays the token game.

6 Conclusion

More and more, system engineering turns out to be the next challenge of engineering. A major problem to be solved in this context is to construct a common modelling base. Such a modelling base has to be applicable not only to technical objects but also to organzational ones like design processes. In this contribution the attempt has been made to show the adequateness of extended (i.e. by hierarchy and recursion) Pr/T-nets to serve as such a common model. By discussing some applications in very different areas the power and flexibility of this modelling concept has been demonstrated. At Cadlab, Paderborn, currently a system engineering environment based on Pr/T-nets is under development.

References

[1] P. Altenbernd, R. Milczewski, "Description of Timing Problems Using Petri-Nets for Level-Independent Timing Verification" *Cadlab External Report*, 7/93, Cadlab, Paderborn, 1993

[2] P. Altenbernd, J. Strathaus, "Solving the Path Sensitization Problem in Linear Time" *Proc. EDAC'92*, 1992

[3] M. Brielmann, B. Kleinjohann, "A Formal Model for coupling computer based systems and physical systems" *Proc. EURO-DAC'93*, 1993

[4] A. Bretschneider, C. Kopf, H. Lagger, A. Hsu, E. Wei, "Knowledge Based Design Flow Management" *IEEE International Conference on Computer-Aided Design* 1990

[5] T.A. Chu, "Synthesis of self-timed control circuits from graphs an example" *Proc. ICCD'86*, 1986

[6] L.A. Cherkasova, V.E. Kotov, "Structured Nets" *Proc. MFCS'81, Springer LNCS 118*, 1981

[7] A. Casotto, A.R. Newton, A. Sangiovanni-Vincentelli, "Design Management Based on Design Traces" *Proc. DAC'90*, 1990

[8] A. DiJanni, "A Monitor for Complex CAD Systems" *Proc. DAC'86*, 1986

[9] R.K. Gupta, G. DeMicheli, "System Synthesis via Hardware-Software Co-Design" CSL Technical Report CSL-TR, Stanford University, 1992

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

[11] D. Harel: Statecharts, "A visual formalism for complex systems" *Science of Computer Programming*, 8, 1987

[12] B. Kleinjohann, R. Milczewski, "Ein einh. form. Modell zur Schnittstellenspez. und HW-Beschreib. (= A Common Formal Model for Interface Specification and Hardware Description)" *Proc. GI/ITG Workshop Formale Methoden zum Entwurf korrekter Systeme*, Karlsruhe University, Fakult. f. Informatik, Technical Report 10/93, 1993

[13] L. Kupitz, J. Tacken, "DECOR - Tightly Integrated Design Control and Observation" *Proc. ICCAD'92*, 1992

[14] D. Kirstein, "TransNet - Ein Interpr. zur mengenth. Transf. hierarch. Petrinetze (TransNet - An interpreter for set-theoretical transformations of hierarchical Petri nets)" *Dipl.Arb. Univ. -GH-Paderborn, FB 17*, 1992

[15] F.J. Rammig, "System Level Design. in: J. Mermet (ed.): Fundamentals and Standards in Hardware Description" *Kluwer*, 1993

[16] M. Rupprecht, "Implementierung und parallele Verarbeitung von Kommunikationssoftware (= Implementation and Parallel Execution of Communication Software)" *Teubner Texte zur Informatik*, Band 2, 1993

[17] B. Steinmüller, "The JESSI-COMMON-FRAME Project - A Project Overview" *Proc. 3rd IFIP International Workshop on Electronic Design Automation Frameworks* (EDAF'92), North Holland, 1992